AFRL-RI-RS-TR-2008-13
Final Technical Report
January 2008

DYNAMIC RECONFIGURATION AND INTEROPERATION IN INFOSPACE COMMUNITIES

BBN Technologies

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2008-13 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
JAMES HANNA Work Unit Manager	JAMES W. CUSACK Chief, Information Systems Division Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JAN 2008	Final	Sep 05 – Nov 07

4. TITLE AND SUBTITLE

DYNAMIC RECONFIGURATION AND INTEROPERATION IN INFOSPACE COMMUNITIES

5a. CONTRACT NUMBER
FA8750-05-C-0267

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Joseph Loyall, Praveen Sharma, Matthew Gillen, Jianming Ye, Richard Schantz

5d. PROJECT NUMBER
ICED

5e. TASK NUMBER
05

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBN Technologies
10 Moulton Street
Cambridge MA 02138-1119

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISE
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2008-13

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-0091

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Under the DynRIIC project, we produced results in two primary areas of QoS management for information spaces: architecture and algorithms. In the architecture area, we conceived, designed, and prototyped a multi-layered QoS management architecture suitable for information spaces. The multi-layered QoS management system works alongside the information manager in an information space and assigns QoS levels and resource allocations, produces and enforces QoS policies, and actuates QoS controls, including resource controls and information shaping.

In the algorithms area, we developed several QoS allocation algorithms that work with the QoS management system, including resource and QoS allocation for single and multiple information spaces. We evaluated the algorithms indicating the relative merits and use of these algorithms in various scenarios, and prototyped some of the algorithms in the QoS decision maker components of the QoS management system.

15. SUBJECT TERMS
Infospace, infosphere, community of interest (COI), composable, dynamic, interoperability, information management

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James Hanna
U	U	U	UL	149	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

Section **Page**

List of Figures

Figure **Page**

List of Tables

Table **Page**

1 Introduction

1.1 Programmatic Data of the Dynamic Reconfiguration and Interoperation in Infospace Communities Project

The Dynamic Reconfiguration and Interoperation in Infospace Communities (DynRIIC) project ran from September 22, 2005 through November 21, 2007 under the AFRL/RISE Infospace Concept Exploration and Development (ICED) program. BBN Technologies was the Prime Contractor, with Dr. Joseph P. Loyall as the Principal Investigator.

The first part of the DynRIIC project ran from September 22, 2005 through October 2, 2006 and concentrated on architecture and prototype development for QoS management capabilities for dynamic information spaces.

The second part of the project commenced on November 22, 2006 as an Engineering Change Proposal (ECP) to our existing contract and ran until November 21, 2007. This second part of DynRIIC concentrated on research, development, and evaluation of QoS allocation algorithms for dynamic, interoperating information spaces.

1.2 Goals of the project

The ICED program [10] included a focus area of *Dynamic Systems Interoperability*, to develop concepts for the integration and interoperation of information in dynamically changing *Communities of Interest (COIs)* [27]. This focus area includes the goals of designing and developing concepts, interfaces, and technologies for the following (illustrated in Figure 1):

- Dynamic integration and reconfiguration when an asset enters a COI, leaves a COI, and moves between COIs.
- Interoperation of assets within a COI – exchange of information between assets within a COI.
- Interoperation/Sharing of assets between COIs – sharing of assets between COIs in a *Community of Action (COA)*.

This dynamic system interoperability must support environments that

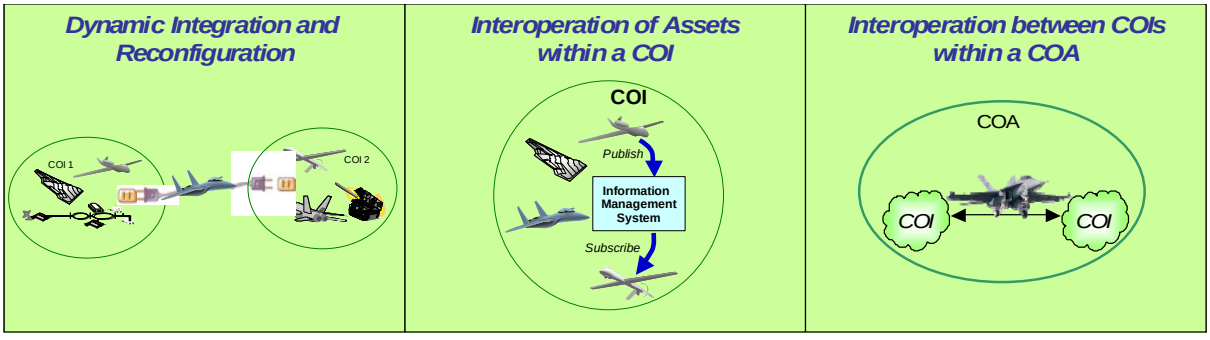


Figure 1: The three primary goals of Dynamic Systems Interoperability

- Change dynamically in terms of a system's mission; the number of participants in the mission; the role each participant plays in, and its relative importance to, the mission; and the available resources.
- Potentially have intermittent connectivity.
- Necessitate information exchange to satisfy mission critical operations.

The concept of *information spaces* has emerged to provide support for the information exchange, brokering, and shared understanding needed by COIs [12]. As information spaces support more and more COIs with time sensitive missions, unfolding situations, and dynamic environments, it is critical to deliver information in real-time with sufficient quality to enable real-time decision making and action. Information delivered too late or with the wrong resolution, form, or precision is insufficient for the user to perform his role in a warfighting scenario. This motivates the need for a *quality of service (QoS)* management capability that can support the dynamic interoperability and real-time requirements of network-centric warfare. In order to be effective, this QoS management capability must work alongside and in conjunction with existing *information management systems (IMSs)* (as illustrated in Figure 2) to manage the production, delivery, and consumption of information within available resources, mediate competing demands for resources, and adjust to dynamic conditions.

An information space QoS management capability must support dynamic resource and information management and reconfiguration as COIs and the makeup of COIs change, in service of critical mission requirements. While providing this service, an infospace QoS management capability should also hide the underlying heterogeneity of the raw resources and communication protocols provided by different platforms.

The main goals of the DynRIIC project were to define and develop concepts for QoS management in dynamic information spaces in the areas of *QoS architecture and design* and *algorithms for QoS management.*

QoS architecture and design – We designed and developed concepts, interfaces, and technologies for dynamic QoS integration and interoperation, at the information space, client, and resource levels. We defined and developed a *QoS Management System (QMS)* that operates

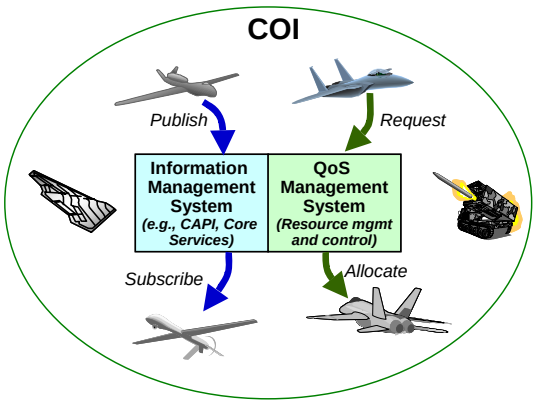


Figure 2: Introducing a QoS management capability alongside the information management capability in a COI

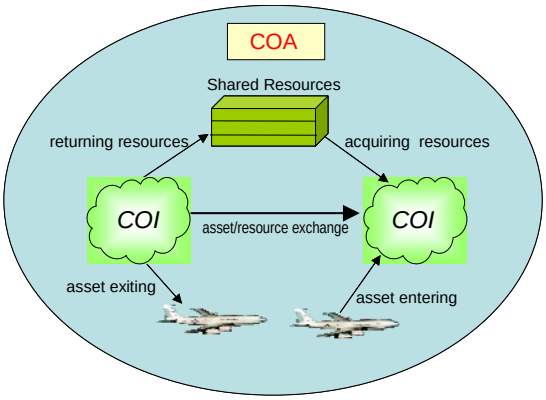


Figure 3: Interoperation of assets between COIs within a COA

alongside the IMS as illustrated in Figure 2 and a set of interfaces for interoperation between the QMS and various IMSs and publication/subscription (pub/sub) services. The interfaces support (1) dynamic integration – assets can unplug from the IMS and QMS of a COI and plug into the IMS and QMS of another COI; (2) Interoperation of assets within a COI using the IMS and QMS of the COI and (3) Interoperation of assets between COIs within a COA utilizing the IMS and QMS of each of the two COIs, as illustrated in Figure 3.

QoS algorithms – We investigated and designed QoS allocation algorithms that consider the complex system dynamics of information spaces, are efficient enough to be used in real-time QoS management, and are scalable to the sizes of envisioned information spaces. This class of algorithms is NP-hard, partially because of the following characteristics:

- There are complex system dynamics among the QoS needs within an information space. That is, how we allocate one resource can impact the demand positively or negatively for other resources.
- There may not be any direct correlation between how important an application is and the amount of resources it needs.
- The resource usage of QoS levels is not monotonically increasing or decreasing. That is, a higher QoS level does not imply more resource usage than a lower QoS level and, in fact, might use more of some resources and fewer of others.
- Resource bottlenecks can change dynamically. That is, addressing a bottleneck caused by a highly constrained resource can result in a bottleneck in another resource.

1.3 Background

The work in this project builds upon the emerging concepts underlying the Joint Battlespace Infosphere [4], including information spaces, communities of interest, and related concepts. This section provides definitions and background information for these concepts as assumed and used in the execution of this project.

A *Community of Interest (COI)* is a collaborative group of *users* or *assets* who must exchange information in pursuit of a set of shared goals, interests, missions, or business processes. The

COI must therefore have a shared vocabulary for the information its members exchange and a shared information exchange mechanism.

An *information space (infospace)* is the cyberspace equivalent of a COI. An information space consists of the *information management system (IMS)* providing information brokering and dissemination; the shared information dictionary understood by users and assets in a COI; interfaces and mechanisms providing access to, and storage of, information; and a set of *clients*, software processes representing the users and assets in a COI.

A *Community of Action (COA) is* a composition of multiple, cooperating COIs. COAs are supported by composed, or *federated*, information spaces.

1.4 Summary of Major Results

Under the DynRIIC project, we produced results in two primary areas of QoS management for information spaces: *architecture* and *algorithms*. In the architecture area, we conceived, designed, and prototyped a multi-layered QoS management architecture suitable for information spaces. The multi-layered QoS management system works alongside the information manager in an information space and assigns QoS levels and resource allocations, produces and enforces QoS policies, and actuates QoS controls, including resource controls and information shaping.

In the algorithms area, we developed several QoS allocation algorithms that work with the QoS management system, including resource and QoS allocation algorithms for single and multiple information spaces. We evaluated the algorithms indicating the relative merits and use of these algorithms in various scenarios, and prototyped some of the algorithms in the QoS decision maker components of the QoS management system.

While we did not concentrate on other areas of QoS management, such as QoS policy, QoS mechanisms, and the capture of QoS requirements, we developed prototype components in these areas. This was, in part, to have a full set of components necessary to evaluate and demonstrate our QoS management results and to establish foundational concepts upon which further work in these areas could be based.

1.5 Organization of This Report

This report is organized into nine main sections.

This section, Section 1, provides background and introduction for the DynRIIC project, including programmatic data, the goals of the project, and a summary of major results.

Section 2 presents the architecture we developed for a QoS management system for information spaces.

Section 3 describes the first prototype QMS system that we developed.

Section 4 describes a set of QoS management algorithms that we developed under the DynRIIC project.

Section 5 presents the results of a set of evaluations and experiments that we conducted to determine the execution time and effectiveness of the QoS management algorithms presented in Section 4.

Section 6 describes the second main prototype software that we developed under this project, an enhanced version of the QoS management decision maker that utilizes the algorithms described in Section 4.

Section 0 describes three of the main demonstrations that we conducted of the software developed under the DynRIIC project.

Section 8 presents some conclusions and recommendations for future research directions.

Finally, Section 9 presents a chronological review of the technical activities under the DynRIIC project, as compiled from the monthly status reports.

2 Architecture of a QoS Management System

One of the main goals of the DynRIIC project was to architect, design, and prototype QoS technologies for dynamic information spaces, including the following:

- Support for dynamic resource and information management and reconfiguration as COIs and COI makeup change.
- QoS management decisions based on critical mission requirements.
- Hiding of the underlying heterogeneity of the raw resources and communication protocols provided by different platforms.

One of the key elements underlying information spaces is an Information Management System, such as the AFRL JBI RI, which provides information management and brokering for clients throughout an information space. We designed and developed a *QoS Management System* that works alongside an IMS in an information space as illustrated in Figure 4. The QMS provides QoS management (including resource management and quality of information management) for the assets within a COI and for assets that are shared between COIs, in the dynamically changing, mission driven environment within which COIs operate.

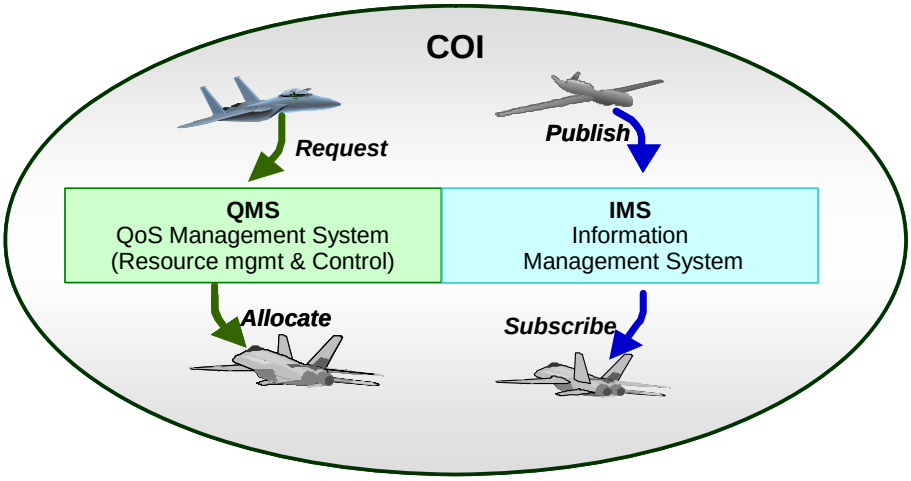


Figure 4: Introducing a QMS alongside the IMS in a COI

The QMS allows entities interoperating within and between COIs to do the following:

- *Specify* their (mission, application, system) requirements and facilitate conversion of these requirements into policies;
- *Measure* the QoS provided in a system;
- *Control* knobs to affect QoS;
- *Adapt* to compensate for changes in QoS; and
- *Mediate* conflicting QoS requirements.

The QMS is designed to support dynamic COIs, in which assets can enter or leave during a COI's operation, and interoperation of COIs, in which assets and resources can be shared between COIs. Assets that enter a COI will join the COI's mission and be assigned a role by a mission manager associated with the COI's command authority. The role the asset plays in the mission determines its importance relative to other assets, and may have an effect on the demands it places for resources within the COI.

The QMS uses a layered management system to manage QoS throughout the COI, as illustrated in Figure 5. *System Resource Managers (SRMs)* provide QoS policies based on the mission requirements and the roles and numbers of assets in the mission, as indicated by the command authority. It pushes these QoS policies to *Local Resource Managers (LRMs)* associated with applications, platforms, or assets, which enforce the policies. The policies determine how many and what kind of resources the asset has access to (such as network bandwidth and computer processing at a command center) and how it should use them to achieve the mission goals (e.g., higher resolution or precision data versus higher rate data). The QMS includes a library of *mechanism components* that access network and CPU controls, and shape data, for enforcing QoS policies.

When assets enter, and are admitted to membership in a COI, the SRM reallocates resources as needed to grant the resources they need. The entering asset requests the resources it needs to perform the role it has been assigned and the SRM determines whether it can provide them without taking resources from higher priority elements. Once the SRM determines the allocation

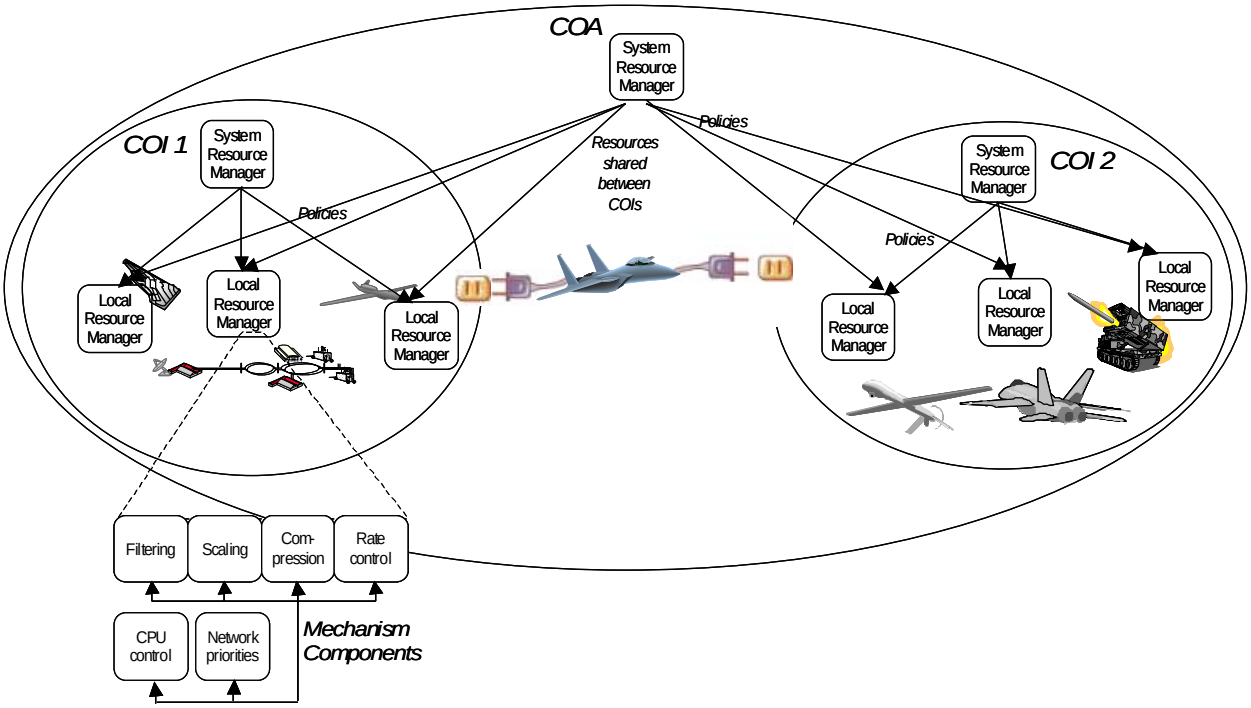


Figure 5: QMS's core components provide dynamic integration and reconfiguration in a COI and dynamic interoperation of assets shared by cooperating COIs.

of resources the asset gets, it is encoded as a contract that the asset's LRM enforces. Assets that do not join a COI or are not admitted to membership in the COI are either treated non-preferentially or are blocked from access to the IMS, depending on the COI's command policy. Assets treated non-preferentially are allowed to publish and receive information and therefore use resources, but only if resources are available and only with "best effort" service.

As illustrated in Figure 6, the QMS architecture consists of core components, a set of interfaces, and IMS adapters. The core components provide dynamic QoS management capabilities. The interfaces facilitate communication between the QMS and clients, and between the QMS and a mission manager, such as a commander, demonstration driver, or other control agent. IMS adapters provide communication between the QMS, clients, and IMSs.

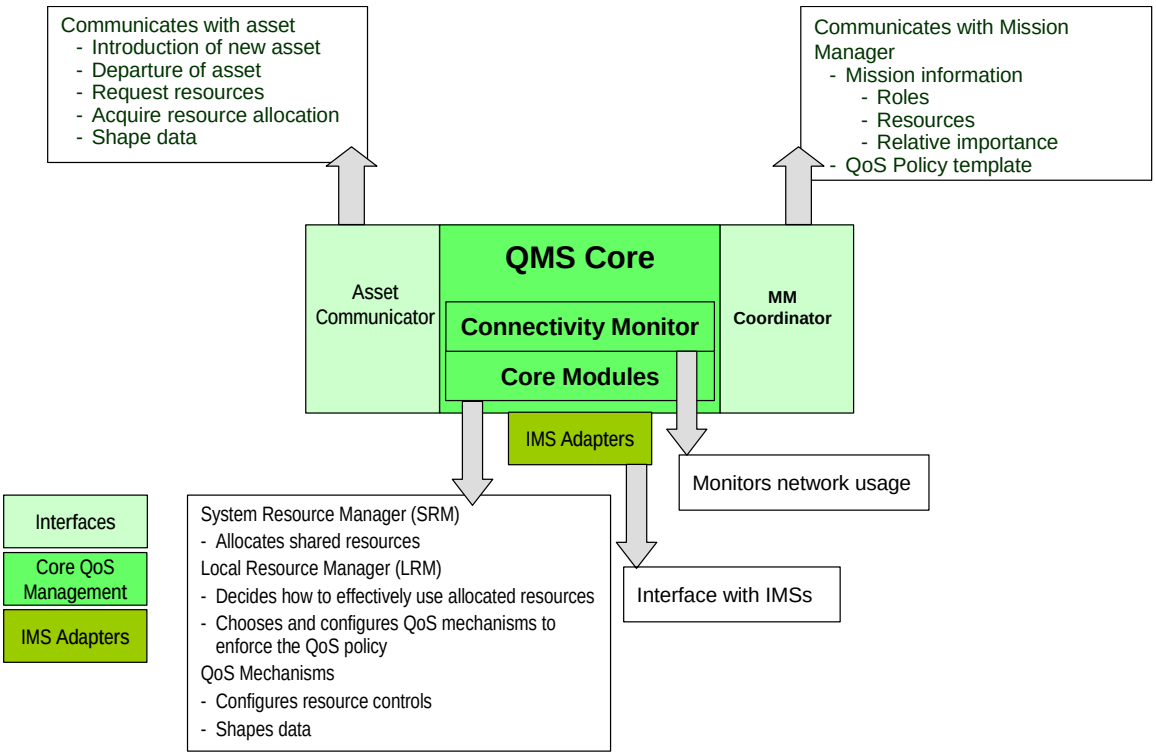


Figure 6: Architecture of QMS

2.1 QMS Core Components

QMS Core consists of core modules and a connectivity monitor. The core modules are illustrated in Figure 7 and include System Resource Managers, Local Resource Managers, and QoS mechanisms. The *System Resource Manager* creates QoS policies based on mission policies, roles, and priorities, and resource availability. The QoS policies include allocations of resources for clients (e.g., an amount of network bandwidth and percent of CPU) and guidance on how clients should use the resources to achieve their mission goals (e.g., for higher rate, higher

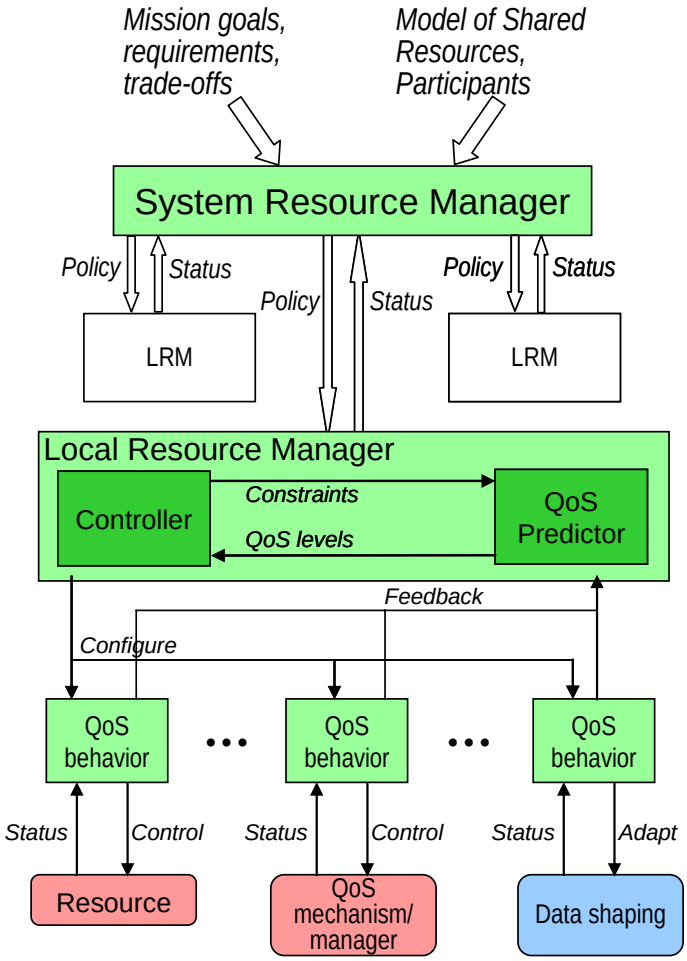


Figure 7: QMS core modules include a layered QoS management architecture of system and local managers and QoS mechanisms

precision information, or higher resolution). The SRM pushes these QoS policies to an LRM associated with each client.

The *Local Resource Manager* enforces the resource allocations provided by the SRM. It selects, configures, and actuates the mechanisms necessary to meet the QoS policies within the resource allocations and provides feedback about actual usage. It also adjusts configurations and mechanisms as needed based on the feedback.

QoS mechanisms actuate specific QoS controls and adaptations. They control resource management interfaces, shape data to provide the desired quality within the allocated resources, and adjust adaptations as needed in response to feedback. Mechanisms can include the following:

- Application adaptations – Altering the rate, size, form, and even content of data.
- CPU controls – Setting process priorities, scheduling parameters, or reserving CPU cycles [6].

- Network controls – Setting bandwidth priorities or traffic classes [9] or bandwidth reservations [28].
- IMS controls – Setting the parameters exposed by IMSs or pub/sub services [18].

The *Connectivity Monitor (CM)* monitors resource (e.g., bandwidth and CPU) usage, the rate of data published into an IMS, processing time of an IMS, the rate at which the subscribers receive the data, and other measures useful in QoS management, reporting, and analysis. As illustrated in Figure 8, a CM consists of *Edge Components* that monitor resource usage on both sides of the network and a *coordinator* that analyzes and reports the information to the QoS Internal Display GUI, described below in Section 3.4.2.

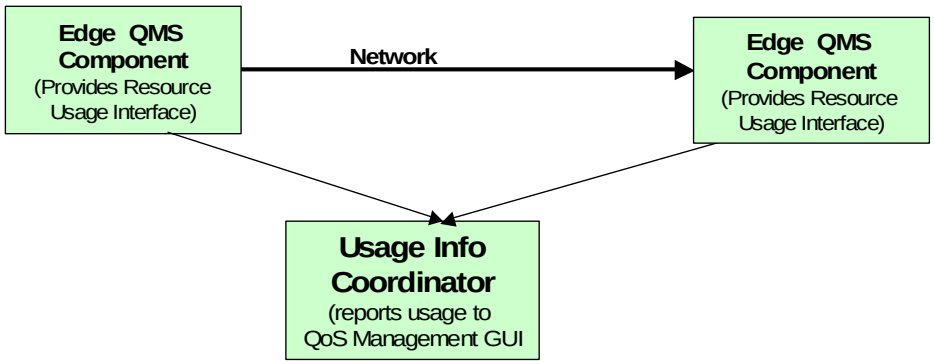


Figure 8: Architecture of the Connectivity Monitor

2.2 QMS Interfaces

The *Asset Communicator (AC)* enables an asset to introduce its capabilities to the SRM and mission manager. This interface includes submitting a membership request when a client first joins an information space.

The *Mission Manager Coordinator (MMC)* facilitates communication between the mission manager and SRM; and between the mission manager and information producing assets/clients. This interface supports the specification of the total available resources in a COI to a SRM, the setting of initial policies, and specifications of roles in a mission. It also enables the SRM to react to events that can trigger resource reallocations, such as changes in roles, priorities, or client activation or deactivation.

The QMS uses *IMS Adapters* to handle the differences between types of available IMSs, including differences in support for sending/receiving information, configuration and QoS options, and implementation languages.

3 The Prototype QoS Management System Version 1

In the first year of the DynRIIC project, we developed version 1 of a QMS that provides the following features:

- Dynamic resource allocation to clients based on mission–driven policies and resource requests
    - In an information space or between interoperating information spaces.
    - On dynamic entry and exit of assets in a COI.
    - On change of mission policies, mission roles, or priority of assets.
- Dynamic and policy-driven QoS management of information being published and subscribed to in IMSs. This QoS management includes rate/latency management, accuracy (such as cropping), precision (such as compression), network management (such as setting of Diffserv codepoints), and CPU management based on the priority, mission, and role of the participating assets.
- Monitoring of resource allocation and usage.
- Support for a variety of IMSs and pub/sub services, including JBI RI 1.2.6, TAO Notification Service [19], and OCI's Data Distribution Service (DDS) [18], OpenDDS 0.8 [20].
- Support software, including a prototype Mission Manager GUI and a QoS Management Internal Display.

The prototype QMS implementation includes an implementation of an SRM and LRM, a set of QoS mechanisms, a connectivity monitor, a structure for QoS policy, and adapters for the JBI RI and DDS. To demonstrate and evaluate the effectiveness of the QMS enhanced QoS managed information delivery, we also developed a prototype application of publisher and consumer clients exchanging real-time imagery data, with a variety of roles and QoS requirements. We also prototyped a mission manager GUI that serves as a demonstration driver, a QoS GUI that displays the internals of the QoS behavior and its relation to the policies, and a prototype membership contract for new clients to enter information spaces in the demonstration.

3.1 Prototype QMS Core Elements

3.1.1 The Prototype System Resource Manager

Our prototype SRM takes as input the number and types of resources available, the number of assets in each role (e.g., surveillance or targeting), the relative weight of each role (e.g., targeting is twice as important as surveillance), and the relative weights of each COI/infospace (e.g., an information space servicing an anti-terror mission is ten times as important as a logistics COI). For each resource shared by multiple clients, the SRM uses the algorithms described in Section 4.1 to determine an allocation of resources for each client, based upon the client's role, relative weights, and the amount of contention for the resource. The SRM compares the resource

allocation to a minimum and maximum allowable and never allocates more than the maximum nor, when possible, less than the minimum.

The minimums and maximums are computed offline for each type of client and role and are based on the minimum quality acceptable for the role and the maximum quality that can currently be achieved. The SRM prototype utilizes an SRM for each information space (i.e., each COI). When resources and/or assets are shared between information spaces, another SRM is instantiated on the fly to allocate the shared resources.

The resource allocation is embedded with the rest of the QoS policy, described below, and distributed to the LRMs for each client using the resource. The prototype supports CORBA IDL and XML structures for storing, populating, and disseminating the QoS policies.

3.1.2 The Prototype Local Resource Manager

The design of the LRM is illustrated in Figure 9 and described in more detail in [15]. The LRM takes as inputs the resource allocations, minimums and maximums for each QoS attribute (e.g., size, rate, compression level, etc.), and the tradeoffs for each role (i.e., which QoS attributes are most important for the role). It computes the appropriate levels for each attribute and sets the proper attributes on the QoS mechanisms. The LRM includes a feedback loop, monitoring the actual resource usage and QoS provided (using the Connectivity Monitor) and adjusting the attribute settings if the resource usage or provided QoS significantly diverges from the QoS policy.

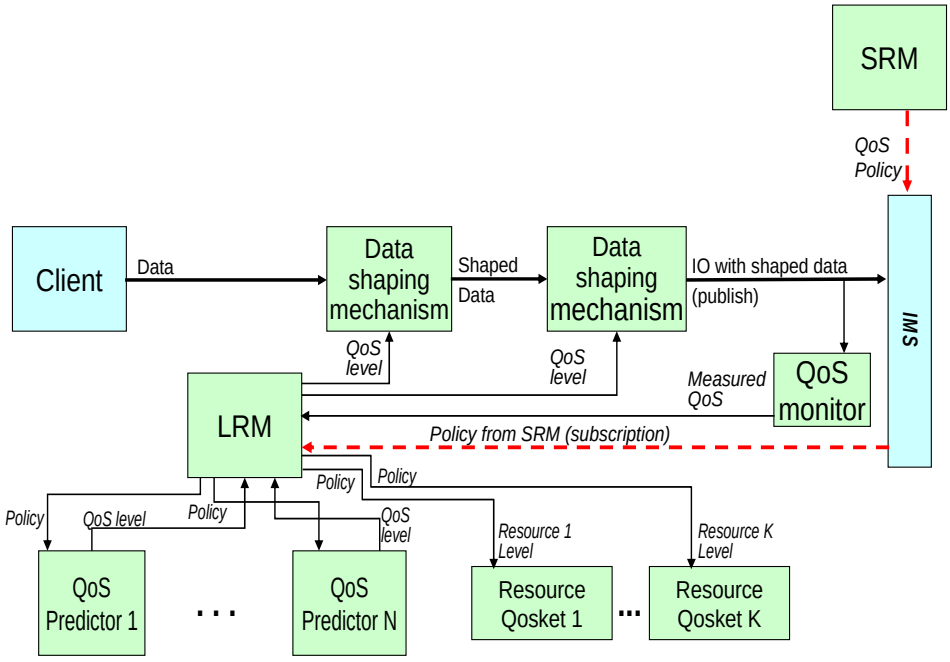


Figure 9: The design of the LRM and its relation to other QMS elements.

3.1.3 QoS Mechanisms

We instantiated QoS mechanisms as software components called *qoskets* [23, 24], including compress, scale, and crop qoskets that shape image information objects; CPU and Diffserv qoskets that set operating system and network priorities, respectively; and control of the rate at which information objects are published. Each QoS mechanism takes as input a level to configure it, which is provided by the LRM dynamically based on the resource allocation, role, and QoS policies.

3.1.4 QoS Policies

The policies include a set of QoS constraints and tradeoffs associated with the specific role a client is playing in a COI. The QoS constraints in our prototype are provided by a default policy created off-line and can be changed manually at runtime using the Mission Manager GUI, described below. The SRM then inserts the resource allocation before sending it to the LRMs for enforcement. Table 1 shows an example of QoS policies for three different roles, including ranges of resources – bandwidth and CPU – that a client can utilize and acceptable ranges for a set of information quality attributes, such as rate, scaling, compression and cropping. We created XML and CORBA IDL structures for storing and passing around policy information.

Table 1: Sample QoS policies for surveillance, target tracking, and battle damage assessment roles.

Mission	Relative Priorities
ISR_COI	**1**
TST_COI	**2**

QoS Constraints and Tradeoffs								
Roles	Relative Priority	Resource Needed			Quality of Information Needed			
		BW Needed (kbps) (Min-Max)	DiffServ Codepoint	CPU (Re-ceiver) (%)	Rate (Timeliness) IO/Frame Rate	Scaling (Size)	Compression (Accuracy)	Cropping (Precision)
SURVEILLANCE (ISR)	1	50-200	NORMAL	0.1-2.0	0.1 – 0.4	Qtr-Qtr	JPG-JPG	None
TARGET TRACKING (TT)	6	150-600	CRITICAL	1.5-5.5	1-1.5	Half-Half	None - JPG	None
BATTLE DAMAGE ASSESSMENT (BDA)	4	300-400	URGENT	1.5-3.0	0.25-0.5	Full-Full	None-JPG	None-30%

3.2 Interfaces

QMS is built using CIAO CCM [5], so all the communications between QMS components are based on CORBA interfaces. The interfaces that QMS uses to communicate with the application

components and IMSs are provided as XML schemas and metadata files, and also as CORBA IDL. The first format is suitable for XML based IMSs, such as the JBI RI, and the second is suitable for CORBA based pub/sub services, such as the Notification Service and DDS.

3.2.1 Asset Communicator (AC)

An asset uses this interface to publish a membership contract to introduce its capabilities to the SRM and mission manager. QMS provides an IDL and an XML version of this interface. The CORBA interface is

```
module uav {
  struct Resource {
    string units;
    double minVal;
    double maxVal;
  };

  struct Resources {
    Resource BW;
    Resource CPU;
  };

  struct InitiateJoin {
    string uavIdentifier;
    string acceptable_roles;
    string parentCOI; //to be filled by SRM
    string sharedByCOIs;//to be filled by SRM
    boolean isActive;
    boolean qosManaged;
    Resources resourcesNeeded; //total resources
    Resources resourcesNeededInSURVEILLANCE; //resources in
specific roles
    Resources resourcesNeededInTARGET_TRACKING;
    Resources resourcesNeededInBATTLE_DAMAGE_ASSESSMENT;
    Resources resourcesProvided;
    string inputDataType;
    string outputDataType;
    string timeOfJoining;
    string leasePeriod; //duration for which this UAV will
be active
    string xml_contract;
  };
};
```

Using this interface, a joining asset identifies itself to SRM by presenting

- its identity (uavIdentifier),
- the roles it can play (acceptable_roles),
- the COI to which it belongs (the SRM fills this information in if the asset doesn't),
- whether it shares resources with other COIs (sharedByCOIs),
- whether it is active (isActive),
- whether it is currently under QoS management,
- the resources it needs (total), and the resources it needs for specific roles,
- the resources it provides,
- the type of data it publishes and type of data it consumes,
- the time at which it joined a COI,
- the leasePeriod and how long it wants to be QoS managed – after this period the SRM assumes the asset to be inactive unless a new lease is submitted.

The field *xml_contract* is the xml version of this contract. It is there to provide flexibility for the SRM to select the interface to use. For instance, if an SRM is using the JBI, it uses the XML based interface. If an SRM is using the CORBA Notification Service, it uses the CORBA interface.

3.2.2 Mission Manager Coordinator (MMC)

The MMC facilitates communication between the Mission Manager and SRM; and between the Mission Manager and assets. Similar to the Asset Communicator, this has both XML and CORBA interfaces.

Communication between the Mission Manager and SRM involves the following interfaces:

- Submitting total resources available to the assets in a COI

```
struct TotalResources {
    string coi_name;
    double avail_bw;
    string bw_units;
    double avail_cpu;
    string cpu_units;
    string xml_contract;
  };
```

- Submitting a QoS Policy template to the SRM. This has two parts – an abstract policy that can be applied to any data:

```
//QoSPolicyTemplate
struct QoSPolicyTemplate {
```

```
    string role;
    string priority;
    string parent_coi;
    ResourcesAllocated BW;
    ResourcesAllocated CPU;
    string rate_units;
    string size_metric; //mapped to scaling for image data
    string size_units;
    string accuracy_metric; //mapped to compression for
image data
    string accuracy_units;
    string precision_metric; //mapped to cropping for image
data
    string precision_units;
    QoSConstraints  qs_S;  //for Surveillance
    QoSConstraints  qs_TT; // for TT
    QoSConstraints  qs_BDI; // for BDA
    string xml_contract;
    //Jitter - only in xml
  };
```

And a policy specific for image data:

```
struct PolicyStatus {
    string uavIdentifier;
    PCES_UAV::Role roleValue;
    PCES_UAV::Priority priorityValue;
    string parentCOI;
    //COIs_Shared coisSharedWith;
    string coisSharedWith;
    double min_frame_rate;
    double max_frame_rate;
    double min_cpu_reservation;
    double max_cpu_reservation;
    long min_compression;
    long max_compression;
    long min_cropping;
    long max_cropping;
    long min_scaling;
    long max_scaling;
  };
```

- Submitting weights

```
struct BW_CPU_Weights {
```

```
    double bw_surveillance_weight;
    double bw_tracking_weight;
    double bw_bda_weight;

    double bw_urgent_priority_weight;
    double bw_high_priority_weight;
    double bw_normal_priority_weight;
    double bw_low_priority_weight;

    double cpu_surveillance_weight;
    double cpu_tracking_weight;
    double cpu_bda_weight;

    double cpu_urgent_priority_weight;
    double cpu_high_priority_weight;
    double cpu_normal_priority_weight;
    double cpu_low_priority_weight;

    //Weights for different missions
    //where mission is represented by a COI
    double isr_mission_weight;
    double tst_mission_weight;
    double bdi_mission_weight;
  };
```

- Submitting Activate Asset command – The Mission Manager instructs an SRM that an asset is active and needs to be QoS managed

```
struct ActivateAsset {
    string uavIdentifier;
    string role;
    string priority;
    string parent_coi;
    boolean active;
  };
```

- Submitting Change Role Command – The Mission Manager instructs an SRM that the role of an asset has changed

```
struct ChangeRole {
    string uavIdentifier;
    string newRole;
  };
```

- Submitting Change COI Command – The Mission Manager instructs an SRM that the COI of an asset has changed

```
struct ChangeCOI {
    string uavIdentifier;
    string newCOI;
  };
```

- Submitting Change Priority Command – The Mission Manager instructs an SRM that the priority of an asset has changed

```
struct ChangePriority {
    string uavIdentifier;
    string newPriority;
  };
```

Communication between the Mission Manager and Asset involves the following interface:

- Start and Stop Signal for assets to send data

```
struct ImageState {
   boolean startProcessing; /** true to begin image and
false to stop image */
   long imageRate;
   Location imageLocation; /** image location where the
asset should move to*/
};
```

3.3 IMS Adapters

QMS uses adapters to communicate with the JBI RI IMS and DDS. We assume that a COI contains at least one IMS or pub/sub service. In our prototype software, we use separate dissemination channels for data traffic (i.e., MIOs) and control traffic (including resource allocations, QoS policies, and mission manager messages).

We use the CORBA Notification Service [19] for the control messages between the QMS and assets, the QMS and the Mission Manager, and between the components of the QMS. The CORBA Notification Service is a channel based event service – publishers publish information on a specific channel and subscribers listen on specific channels. The subscribers are notified if the status of the channels changes, e.g., if new messages arrive on the channel. The messages can be further filtered based on the topic or event content titles.

For data traffic, we prototyped and demonstrated use of the JBI RI IMS and DDS.

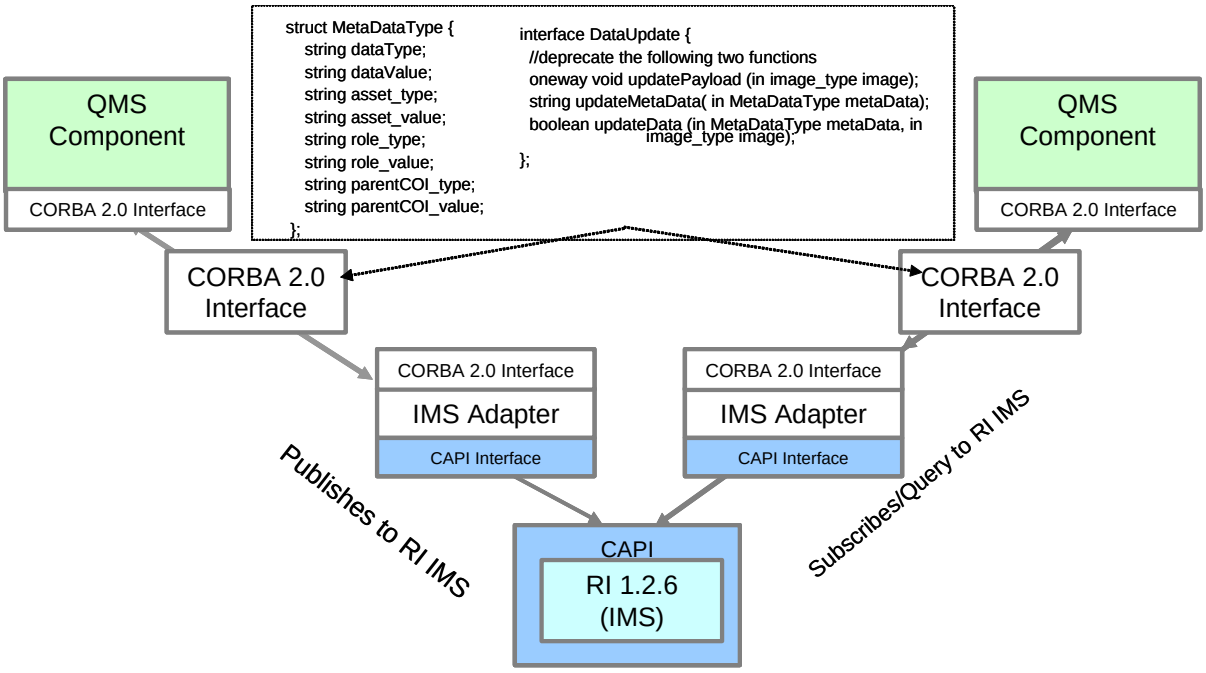


Figure 10: Pattern of communication between QMS components and the JBI RI IMS

3.3.1 Communication with the JBI RI IMS

Because of the heterogeneity of the languages and component models (QMS is primarily written in the C++ implementation of CCM, i.e., CORBA 3.0 [17], and the JBI RI 1.2.6 is written in Java) used in the QMS and IMS, we provided adapters for them to communicate. Since CORBA 3.0 is a superset of CORBA 2.0, it is not a straightforward process for a CORBA 2.0 interface to invoke CCM components. To overcome this, QMS provides CORBA 2.0 interfaces that can invoke any other CORBA 2.0 interfaces, and vice versa. The adapters provide CORBA 2.0 interfaces to communicate with QMS and Java interfaces to communicate with the Common API (CAPI) of the JBI RI. There needs to be two adapter components – one publisher to the IMS and one subscriber/query from the IMS – for every QMS component that interacts with the IMS, specifically the SRM, LRM and Edge Components.

Figure 10 illustrates the communication pattern between the QMS components and the JBI RI. An alternative to this pattern would be to use JNI (Java Native Interface). However, using CCM with JNI increases the complexity of communication (requiring marshalling and demarshalling a CCM event or facet/receptacle call) and increases the latency in QMS response time (because it entails loading a Java virtual machine into a CCM process).

3.3.2 Communication with DDS

Implementations of DDS vary. We integrated and tested with OpenDDS from OCI, an open-source version of DDS based on CORBA [20]. Because OpenDDS uses CORBA 2.0 (whereas

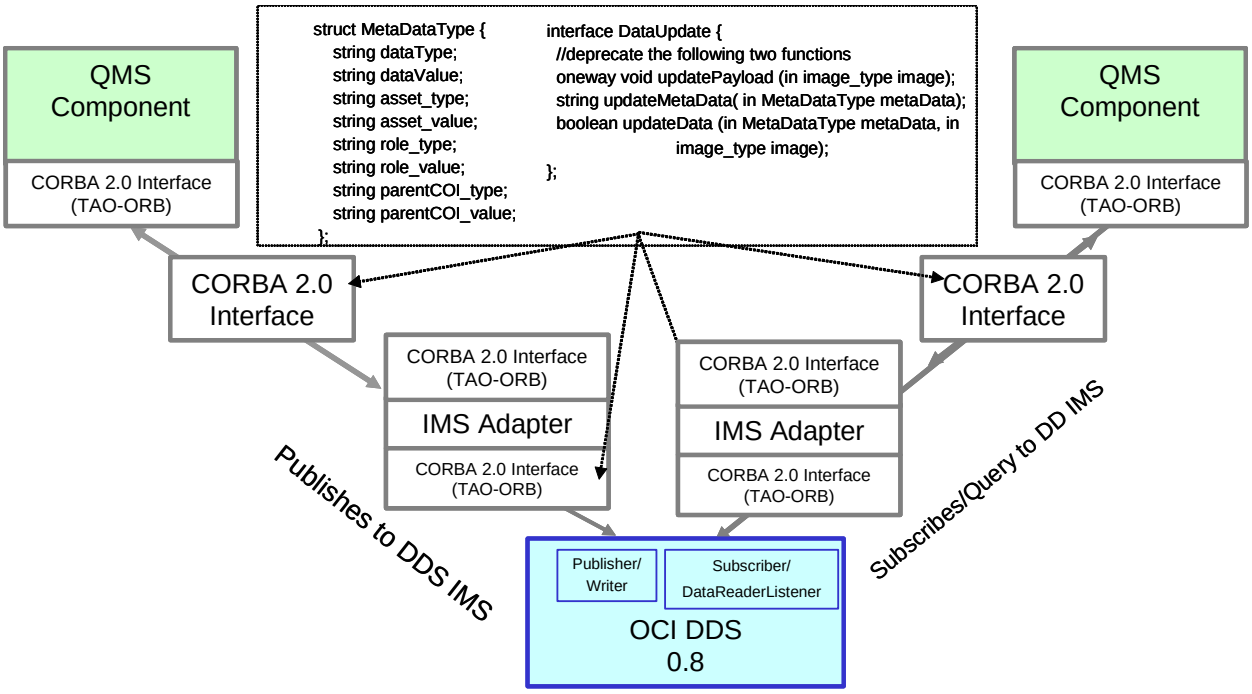


Figure 11: Pattern of communication between QMS components and DDS

the QMS uses CORBA 3.0) and to maintain uniformity with the QMS interactions with the JBI RI, the QMS software also provides adapters for DDS. The QMS adapter for DDS is shown in Figure 11.

3.4 Prototype Support Software for the QoS Management System

Although not a focus of our research and development, we also prototyped instances of support software for the QMS system, including a *Mission Manager* for entering mission-level QoS requirements and driving demonstrations, a *QoS Internals* display showing the behavior of the system under QoS management, and sample clients.

3.4.1 Mission Manager GUI

One of the key aspects of the QMS concept, design, and instantiation is that it is mission-driven. That is, it makes its QoS decisions based on an understanding of the mission or goals of the collection of distributed applications making up information spaces, the roles clients are playing in the overall mission, and the relative importance of each client and role. While we have described missions in defense terms, the concept applies to any domain in which a distributed system has a set of overarching goals. For example, a military mission might be rapid and effective response to time sensitive threats; a commerce mission might be to maximize sales; a critical infrastructure mission might be delivery of commodities such as power to users; and an emergency response mission might be the peacetime equivalent of time sensitive targeting, i.e.,

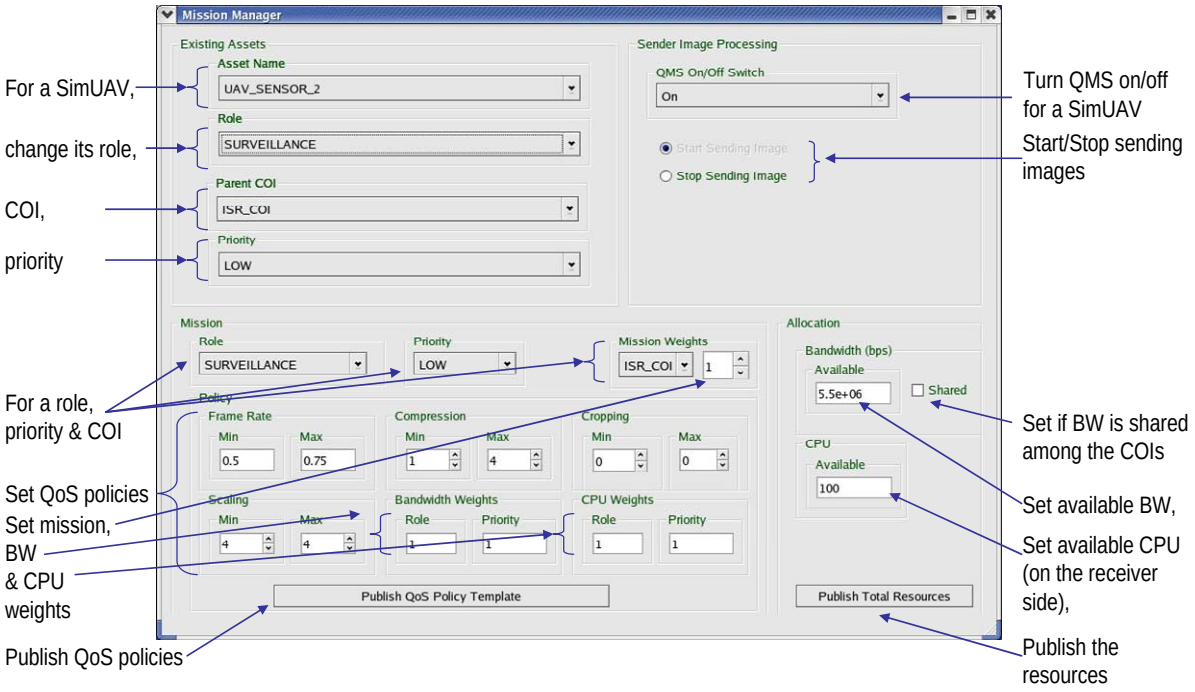


Figure 12: Prototype Mission Manager GUI

rapid and effective response to emergency and disaster situations. The role of a mission manager can be played by a human command authority or by a scripted demonstration driver. Regardless, the need for command and control tools to support mission management is clear. Therefore, we developed a prototype Mission Manager GUI, simulating a command authority that drives a mission in network-centric warfare and enabling control of the QMS and applications using it. As illustrated in Figure 12, the Mission Manager GUI enables a user to start and stop clients, change the roles and priorities of clients, move clients between COIs (and therefore information spaces), set the ranges for QoS policies, and indicate the amount of available resources and whether they are shared.

3.4.2 QoS Internals Display

To better support testing, demonstrating, and measuring the benefits of the QMS, we developed a prototype graphical display of the QoS policies, delivered QoS, resource allocations, and actual resource usage, depicted in Figure 13. The information displayed on this console is gathered in several places in the QMS prototype, including by the Connectivity Controller, SRMs, LRMs, and Mission Manager. The information is updated in real-time and published to the CORBA Notification Service, to which the QoS Display subscribes. The QoS Display has four main subparts:

- *Client identification information* which identifies the client, its role, priority, the COI to which it belongs, the IMS being used by that COI's information space, and the SRM that manages QoS for this client.

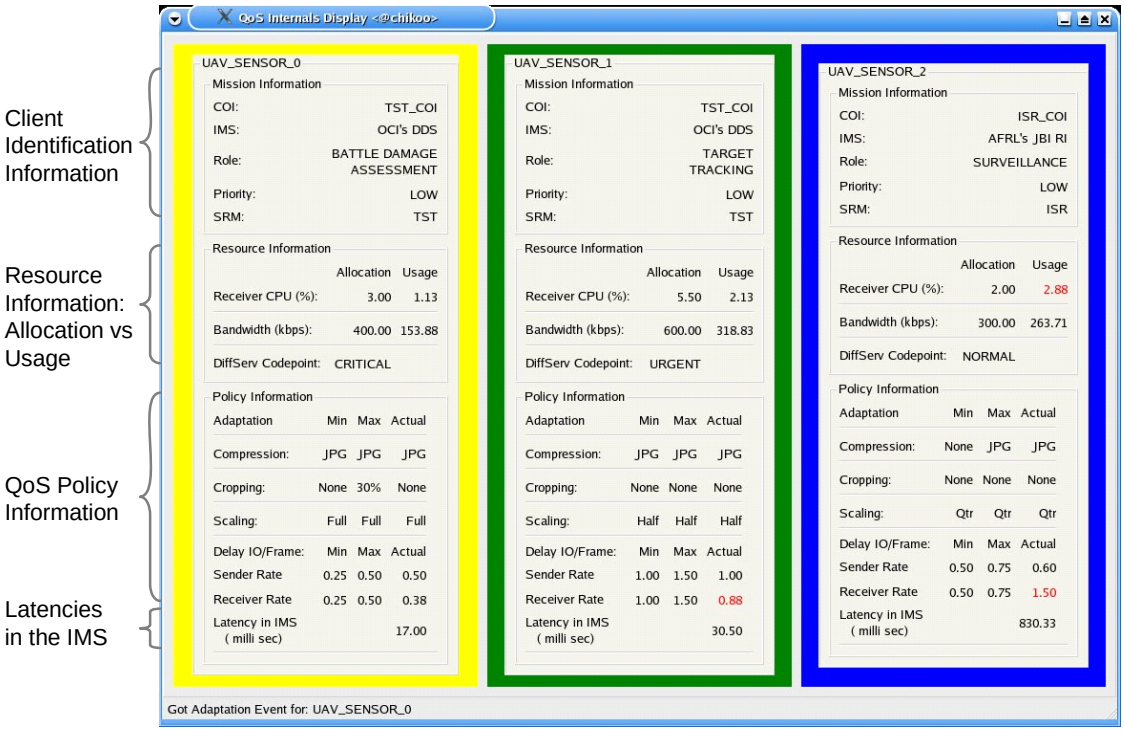


Figure 13: The prototype QoS Internals Display

- *Resource information* which shows the bandwidth and CPU allocated to and used by the client, as well as the network priority (i.e., *Diffserv codepoint*) assigned to traffic originating from the client.
- *QoS policy information* includes the policies for information quality, i.e., the minimum and maximum values for information quality attributes, and the actual values for those attributes at runtime.
- *Latencies in the IMS* displays the time taken for information objects published to the information space's IMS to be processed and delivered to clients subscribing to them.

The GUI uses color coding to convey roles and adherence to policy. For example, in Figure 13, a client performing surveillance (the ISR role) is in a pane with a blue frame; the client performing target tracking (the TT role) is in a green frame; and the client collecting battle damage information (the BDA role) is in a yellow frame. Usage information that adheres to the QoS policies is displayed as black text, while usage information that violates QoS policies and resource allocations is displayed in red.

3.4.3 Sample Clients

To serve as a test and demonstration application, we developed clients simulating imagery sensors and command and control nodes. The imagery sensors are simulated unmanned aerial vehicles (UAVs) similar to those that we used in a flight demonstration at White Sands Missile

Range in 2005 [13]. They read prerecorded UAV imagery gathered by an Electro-Optical (EO) sensor and publish it to an information space. Like the actual UAVs in the flight demonstration, the simulated UAVs can operate in three roles: ISR for surveillance of an area of interest, TT for tracking a target, and BDA for collecting battle damage imagery.

Paired with each imagery publisher is an information receiver that subscribes to the imagery and simulates part of the command and control (C2) functionality in the flight demonstration. The receiver clients include a GUI to display each image as it is received. The GUI, illustrated in Figure 14, frames the imagery display in a color coded frame corresponding to its role (blue for ISR, green for TT, and yellow for BDA). The display provides a visual indication of the QoS management in effect with apparent changes in imagery rate and size when roles change (resolution changes due to compression levels are less visually apparent).

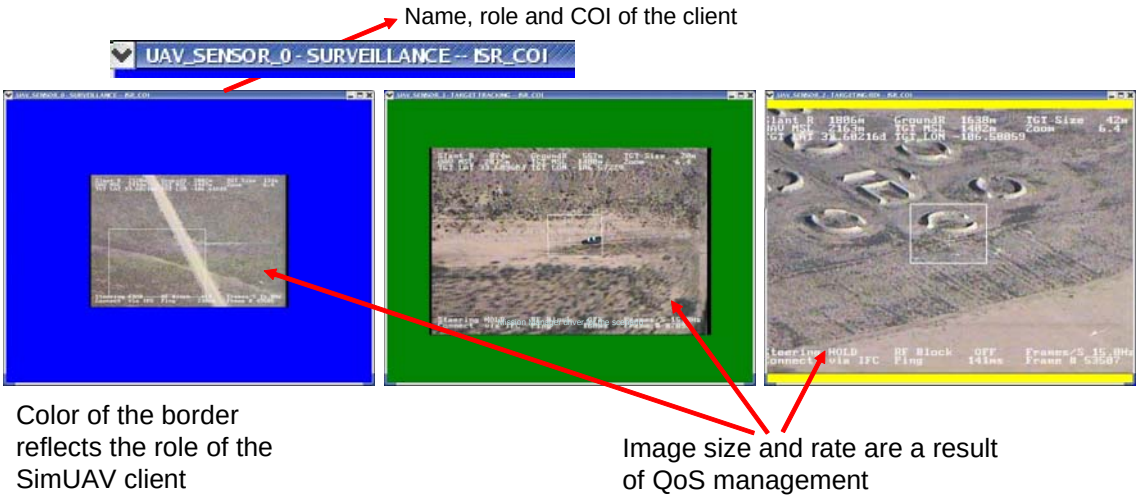


Figure 14: GUI of simulated C2 receiver clients displaying imagery published by simulated UAV clients.

4 QoS Management Algorithms

As described in Section 1.1, the first part of the DynRIIC project concentrated on the architecture and prototyping of QoS management techniques for dynamic information spaces. This included prototyping an initial QoS management algorithm for the SRM component. In the second part of the project, we built upon this basis to define, implement, and evaluate a set of more capable QoS management algorithms for dynamic, interoperating information spaces.

The initial prototype algorithm defined during the first part of the project is a resource allocation algorithm that divides each shared resource among the users based on their relative importance and resource needs.

Since the initial algorithm considered only one resource at a time, it did not consider the system dynamics in its allocations, i.e., the way managing one resource affected others. Therefore, in the second part of the DynRIIC project, we designed and implemented QoS management algorithms that consider multiple QoS levels and the resources needed by each QoS level. These algorithms, called *Multi-Resource Multi-QoS (MRMQ)* allocation algorithms, consider the QoS levels at each place that QoS can be affected (called *control points* or *applications*[1], terms that we use interchangeably), attempting to maximize a measure of overall benefit (i.e., a *utility function*) defined for an information space, within the available resources.

Multi-Resource Multi-QoS allocation is an NP Hard problem [11], leading to a tension between the speed to produce a QoS allocation and the quality of the allocation (in terms of how close to an optimal allocation). One can guarantee arriving at an optimal solution if and only if one examines a search space of all possible allocations, an exponential search in general. Since this is infeasible for all but modestly sized information spaces, we took two simultaneous approaches: (1) developing heuristics that enable the search space to be reduced in many cases, and (2) developing approximation algorithms that run in less than exponential time in the worst case.

We developed three MRMQ algorithms for QoS allocations within an information space, called *intra-information space algorithms*:

- *Optimizing Brute Force* which always provides an optimal solution but takes exponential time in the worst case. We developed two optimizations that can prune the search space and reduce the runtime significantly.
- *Greedy Approximation* is an approximation algorithm based on 0-1 integer programming [26]. This algorithm produces a near optimal allocation in many scenarios and runs in polynomial time.
- *Multi-Resource Multi-QoS Knapsack Approximation (MMKP)* is a dynamic programming approach in which we extended existing multi-dimensional knapsack approaches [16] to include the third dimension of multiple resources.

Since one of the primary goals of the ICED program, and hence the DynRIIC project, is to support interoperating information spaces, we didn't stop at QoS allocation algorithms for

[1] The algorithms treat related control points, such as those for a single application, as a single control point.

individual information spaces. We also researched algorithms for allocating QoS for applications that share resources across information spaces. We developed a set of *two-phase QoS allocation algorithms*. The first phase determines the applications that share resources across multiple information spaces and divides the shared resources between the information spaces. The resource allocations from the first phase are used as constraints by the second phase, which runs an intra-information space algorithm to allocate QoS within each information space.

This section describes the QoS management algorithms that we developed under DynRIIC. We begin by describing the algorithm used in the initial prototype. Next, we describe each of the three intra-information space algorithms. Finally, we describe the two-phased multi-information space algorithms.

4.1 Resource Management Algorithm

The initial prototype SRM uses a resource allocation algorithm. The algorithm allocates resources based on the following inputs and context:

- The relative importance of the COI's mission compared to other COIs' missions.
- The relative importance of the asset's role compared to other roles in a mission.
- The total amount of resources available.
- The total number of assets sharing a particular resource.

The algorithm iterates over each of the shared resources and calculates an *allocation unit* (*alloc_unit)* based on the following formula:

$$alloc\_unit = \frac{(resources\ available)}{\left(\sum_{i \in roles} (number\ of\ assets\ in\ role\ i) \times (weight\ of\ role\ i)\right)\left(\sum_{j \in COIs} (weight\ of\ COI\ j)\right)} \quad (1)$$

The formula then allocates a number of allocation units to each participant (assets, clients, control points) using a shared resource equal to the weight of its role (i.e., its importance relative to other roles) times the weight of its COI (i.e., the COI's importance relative to that of other COIs), as calculated by the following formula:

$$allocation_{asset\ k} = alloc\_unit \times \left(\sum_{i \in roles} asset_k\ in\ role\ i \Rightarrow weight\ of\ role\ i\right) \times \left(\sum_{j \in COIs} asset_k\ in\ COI\ j \Rightarrow weight\ of\ COI\ j\right) \quad (2)$$

The algorithm is constrained by the following upper and lower bounds:

- It will not allocate more than the maximum amount of a resource that the asset can use.
- It will attempt to not allocate less than the minimum amount of a resource needed by the asset to do something meaningful.

4.2 The Optimizing Brute Force QoS Management Algorithm

The goal of each MRMQ algorithm is to select an *allocation* of applications running at particular QoS levels that simultaneously:

- Is *feasible*, i.e., fits within the resources available in the system (information space or set of information spaces). An infeasible allocation cannot be deployed and hence is not an acceptable solution for the algorithm.
- Maximizes mission *utility*, i.e., allocates the applications of most importance to the overall mission and provides higher QoS where it is most useful to the mission.

While the best *utility function* to use can differ for given situations, missions, or domains, a reasonable utility function to use for information spaces is one that calculates utility based on the criticality of the applications that are run and the QoS level at which they are run. This is captured by the following equation:

$$Utility = \sum_{i=1}^{A} (w_c C_i)(w_q Q_i) \tag{3}$$

where:

- C_i is a measure of the relative criticality of application i compared to other applications.
- Q_i is a value assigned to the QoS level chosen for application i.
- w_c and w_q are weighting factors (to control the tradeoff of running more applications or applications at higher QoS levels).

The feasible allocation with the highest utility is considered the *optimal* allocation, or solution. Notice that there could be multiple allocations with equal utilities, so there could be multiple optimal solutions.

The above utility function does not consider resource efficiency. However, keeping resources in reserve could lead to more effective QoS management in dynamic information spaces because wholesale reconfigurations will be reduced if there are resources available to handle overload situations or the addition of new applications. We can accomplish this by adding a *slack factor* to the utility function, i.e., a numerical measure of the resources available.

$$Utility = \left(\sum_{i=1}^{A} (w_c C_i)(w_q Q_i) \right) + w_r R \tag{4}$$

where:

- R is a measure of the resources available
- w_r is a weighting factor to control the tradeoff of using available resources now to run more applications (or higher QoS levels) or keeping the resources in reserve.

The Brute Force algorithm always provides an optimal solution but potentially runs in exponential time. The algorithm uses the total number of control points and their QoS levels to build a combinatorial decision tree. As depicted in Figure 15, levels of the decision tree represent applications, branches represent QoS levels, and leaf nodes represent combinations of an entire set of applications and QoS levels in an information space.

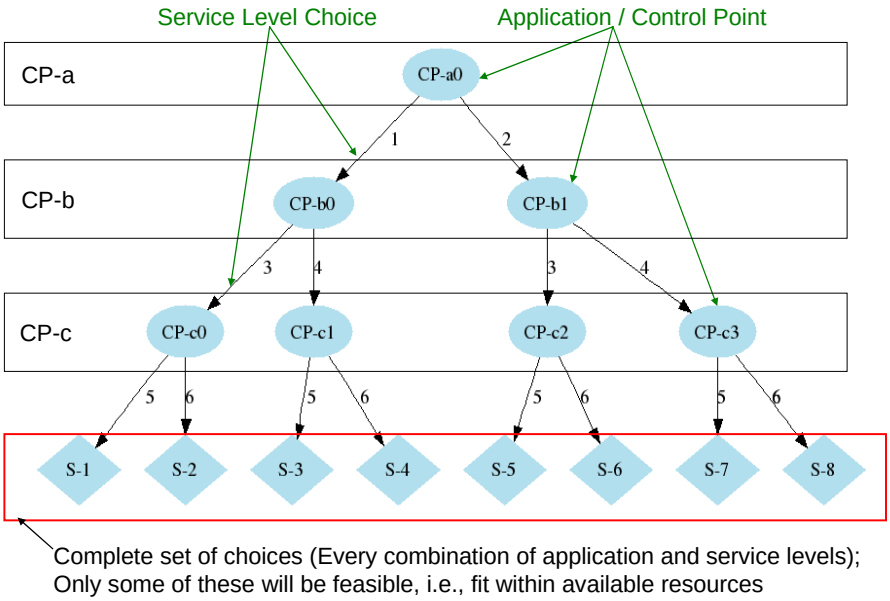


Figure 15: Decision tree that the Optimizing Brute Force algorithm creates and traverses to allocate resources.

The Brute Force algorithm traverses the tree recursively and examines each leaf node for *feasibility* and utility. If the algorithm finds a node to be feasible, the algorithm compares the utility of the node with the highest utility of previously evaluated feasible solutions. If the utility of the node is higher, then this solution becomes the new best solution. The best solution that the algorithm reaches after evaluating all the leaf nodes is the optimal allocation, i.e., the feasible solution with the highest utility.

The basic Brute Force algorithm, with no optimizations, runs in $\Theta(q^a)$ where[2]:

- q is number of QoS levels for each application, and
- a is the number of applications.

We developed two optimizations that can prune the search space, significantly in some cases. The following sections describe these optimizations.

[2] Θ notation is a tight upper and lower bound on the algorithm execution, i.e., the algorithm will check every node of the tree, exactly q^a *nodes.*

4.2.1 Pruning Using an Infeasibility Check

This optimization utilizes the fact that as the algorithm traverses down from the root node to leaf nodes, the number of applications and QoS levels represented in the nodes increases. Consequently if the partial allocation represented by any non-leaf node is not feasible (i.e., it requests more resources than that are available), then all the nodes in the subtree under the non-leaf node are also infeasible (because each will add applications to the already infeasible partial allocation). The entire subtree can be bypassed.

As illustrated in Figure 16, the algorithm analyzes the non-leaf nodes for feasibility as it traverses the decision tree. If at any time, it finds a non-leaf node that fails the feasibility-check, the algorithm prunes the entire subtree under that node. This optimization works well when many of the leaf nodes represent infeasible allocations, leading to significant pruning.

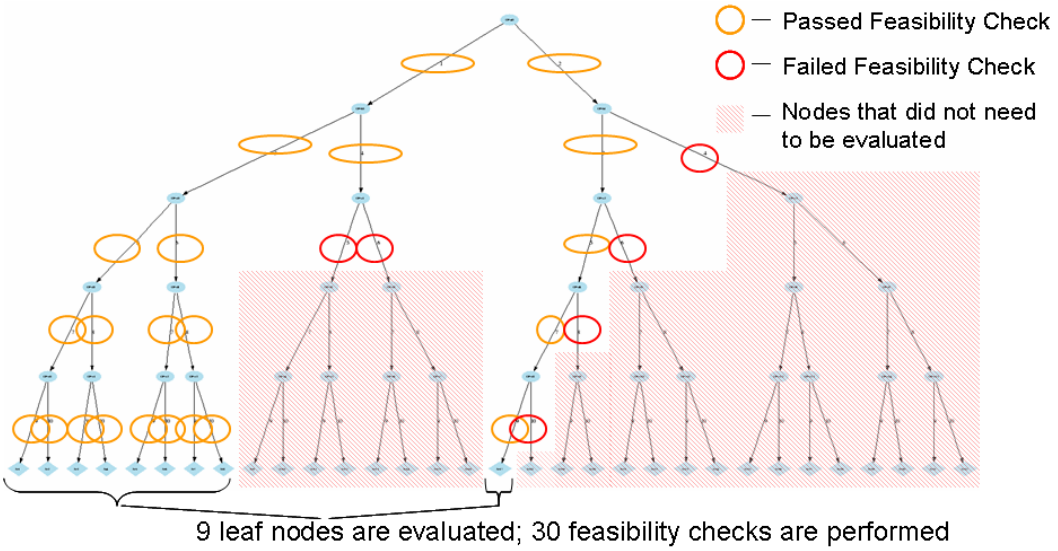


Figure 16: Brute Force using pruning with an infeasibility check.

4.2.2 Pruning Using a Utility Check

This optimization utilizes the fact that as the algorithm traverses down from the root node to leaf nodes (increase in depth of a tree), the number of applications and QoS levels represented in the nodes increases and the utility associated with each node will be more than that of its parent node. As illustrated in Figure 17, at each point in the traversal of the tree, the algorithm walks the path of highest utility first (essentially following the branches of highest QoS levels). If the leaf node reached is lower utility than the best solution reached so far, the entire subtree can be pruned, since all other paths would lead to even lower utility.

This optimization works well when the algorithm finds a high utility feasible solution early, enabling pruning of many subtrees with lesser utility. In the worst case, the algorithm finds many

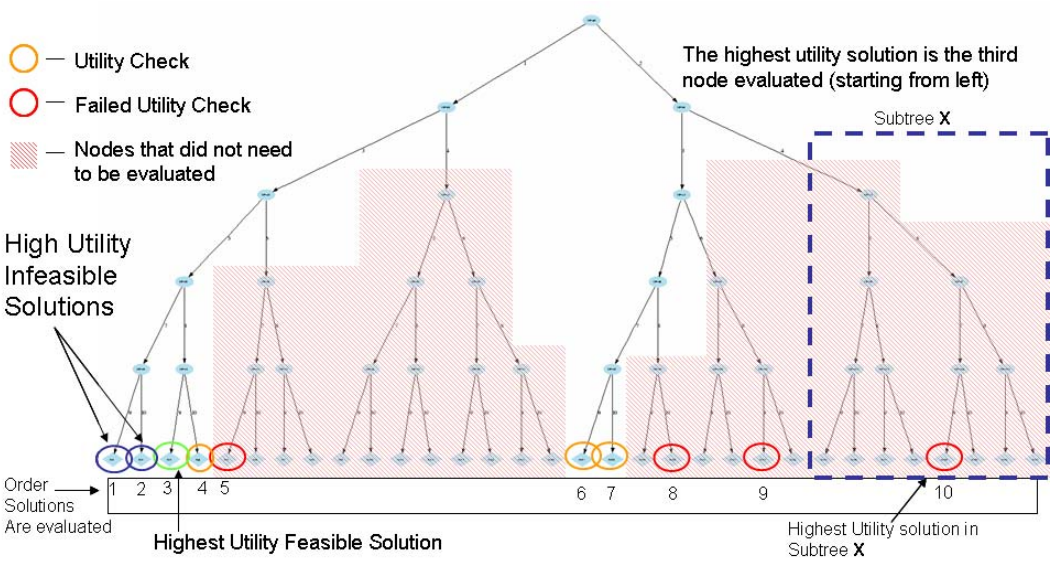


Figure 17: Brute Force using pruning with a utility check.

feasible nodes and relatively low utility solutions, resulting in little or no pruning. In these cases, the algorithm may still end up examining nearly the entire tree. Therefore, the Optimizing Brute Force algorithm is $O(q^a)$.

4.3 The Greedy Approximation QoS Management Algorithm

Greedy Approximation is an approximation algorithm based on 0-1 integer programming. 0-1 integer programming tries to maximize the *objective function*:

$$\sum_{i=1}^{m} p_i x_i \tag{5}$$

subject to

$$\sum_{i=1}^{m} H_{ij} x_i \le L_j \tag{6}$$

for $j = 1, 2, \ldots, n$, where:

- Each x_i is an application at a particular QoS level
- p_i is the priority of the application
- H_{ij} is the resource usage of x_i
- L_j is the vector of the capacity of the resources
- m is the number of applications and n is the number of resources.

Greedy Approximation is based upon an algorithm in [26]. It allocates QoS levels to applications contending for resources greedily using an *effective gradient* measure, a ratio of the benefit that each application provides and the cost that it incurs. The algorithm measures the benefit of an application as the value it contributes to the objective function above. It measures the cost of an application as the amount of resources requested by the application. The algorithm aggregates the resources into a single dimension and assigns a *penalty* to increase the cost associated with requesting a highly contended resource (i.e., a resource for which a significant amount has already been allocated to other applications).

Our algorithm extends the algorithm in [26] in the following ways:

- We have two variable dimensions that need to be considered. Each application can have multiple QoS levels from which to choose. In the algorithm in [26], each application has one service level. Our algorithm treats each combination of application and QoS level as a separate viable choice, while ensuring that only one QoS level can be chosen for each application.
- We compute an initial penalty vector for research usage. The algorithm in [26] only computes penalties as the algorithm progresses, which can lead to significantly suboptimal allocation. That is, it treats all resources equally and completely available at the beginning. In reality, some resources are more likely to become bottlenecks (e.g., because more applications request them or applications request higher amounts of them) than others. Our algorithm performs an initial pass and assigns an initial penalty to highly contended resources, making it cost more to request these resources.

After computing the initial penalty, our algorithm computes the total number of applications (i.e., application-QoS level combination) as described in 1 above. It iterates over the following steps until either all the applications are granted the resources that they request or there are no more resources left to allocate to any remaining applications:

1. It computes the effective gradient for each application as the ratio of benefit divided by cost. The benefit is the utility that a given application at a given QoS level provides. The cost is the resources requested adjusted by the penalty.
2. It selects the application and QoS level combination with the highest effective gradient and eliminates further consideration of the other QoS levels for this application.
3. It allocates the resources needed by the application and QoS level combination selected in step 2, removing those resources from the available resources.
4. It prunes the list of applications of any infeasible choices.

The runtime of the Greedy Approximation algorithm is $O(a^2qr)$, where:

- a is the number of applications,
- q is the number of service levels, and
- r is the number of resources

4.4 The Multi-Resource Multi-QoS Knapsack Approximation QoS Management Algorithm

Multi-Resource Multi-QoS Knapsack Approximation (MMKP) takes a dynamic programming approach to the Multi-dimensional Multi-Constraint 0-1 Knapsack problem. Multi-dimension refers to multiple applications and multi-constraint refers to the multiple QoS levels in which different applications operate. We extended the algorithm to incorporate several different sets of resources instead of handling a single resource. In general, the runtime of this class of the problem is pseudo-polynomial [7], so we developed an approximation algorithm that runs in linear time.

The algorithm tries to maximize a benefit function B:

$$B = \sum b_i \tag{7}$$

where b_i is the benefit of running the ith application and QoS level combination, with the constraint that the sum of all resources used is less than the resources available.

$$\sum r_i \le R \tag{8}$$

The algorithm is a recursive algorithm:

$$B[i, r] = \begin{cases} \text{if } r_i <= r & \max (B[i-1, r], B[i-1, r-ri] + b_i) \\ \text{else} & B[i-1, r] \end{cases}$$

That is, at any step when there are sufficient resources to run the ith application and QoS level combination, that application and QoS level are chosen only if they increase the utility more than using the resources for other choices. If there are not sufficient resources, then the ith application and QoS level combination cannot be chosen.

The algorithm extends MMKP to include multiple resources by normalizing the resource usage of all the resources to a scale of 0 to 1. It then iterates over the resources and applications and selects the QoS level that satisfies the most constrained resource. Since the algorithm is based upon dynamic programming, it uses a bottom-up approach to build a table (memoization) of solved problems, and uses these problems to solve the larger-size problems.

Dynamic programming solutions are pseudo-polynomial in time. As explained in [8]:

> "A *pseudo-polynomial-time* algorithm is one that runs in time polynomial in the dimension of the problem and the *magnitudes* of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes. Such algorithms are technically exponential functions of their input size and are therefore not considered polynomial. Indeed, some *NP*-complete and *NP*-hard problems are pseudo-polynomially solvable (sometimes these are called *weakly NP*-hard or-complete, or *NP*-complete *in the ordinary sense*). For example, the *NP*-hard knapsack problem can be solved by a dynamic programming algorithm requiring a number of steps polynomial in the size of the knapsack and the number of items (assuming that all data are

scaled to be integers). This algorithm is exponential-time since the input sizes of the objects and knapsack are logarithmic in their magnitudes."

In an attempt to develop a polynomial time, or better, version of the dynamic programming algorithm, we had to limit the sizes of the resources by normalizing them to a capacity of 0 to 1 and choosing a quantization, i.e., a discrete unit of allocation for each resource. This results in resources being allocated in discrete units (e.g., tenths, hundredths, or thousandths) and makes the runtime of the algorithm linear, O(a*q*r), where the quantization affects the constant.

For example, a quantization of 0.1 would allocate resources in tenths of their total amount available. This is a coarse grain allocation, e.g., if an application and QoS level requests 3% of a resource, it will get either 0% or 10%. The quantization also places a limit on the number of applications that can share a resource. For example, a 0.1 quantization means that at most ten applications can share any resource.

The quantization provides an important configuration choice for MMKP. A finer grain quantization should, in theory, improve the optimality of the solutions but will increase the runtime significantly. For example, a quantization of 0.01 will allow up to 100 applications to share each resource and will allocate resources in hundredths, but would increase the execution time of the algorithm by at least 10x over that for a 0.1 quantization. For some resources, this would still be a gross quantization. For example, a 100 M link would be allocated in units of 1 M and a 1 G network link would be allocated in units of 10 M. For our baseline MMKP algorithm, evaluated in Section 0, we used a quantization of 0.1.

4.5 Two-Phased Multi-Information Space QoS Allocation Algorithms

We developed a two-phased approach to QoS allocation for multiple information spaces that might share resources. The first phase runs an inter-information space algorithm to identify the resources shared between information spaces and divide them between the information spaces. The resources allocated by the first phase become the total resources available to each information space for the second phase. An intra-information space algorithm is then run in each information space (constrained by the results of the first phase) to allocate QoS to the applications in that information space. The second phase algorithms can be run in parallel, each in their respective information space.

4.5.1 First Phase Inter-Information Space Algorithms

We developed three inter-information space algorithm choices, described in the following sections.

4.5.1.1 Dynamic Approximation Algorithm

The *Dynamic Approximation Algorithm* is a modified version of the G*reedy Approximation* algorithm we described in Section 4.3. First, the algorithm takes an initial pass over the applications in all of the information spaces to discover those that share resources between information spaces. This initial pass takes *O(ar + aqr)*, where:

- a = the total number of applications in all the information spaces,
- r = the total number of resources,
- q = the number of QoS levels for each application

After this initial pass, the algorithm runs the approximation algorithm on the subset of applications that share resources between the information spaces. This takes a'^2qr time, where:

- a' = the number of applications that share resources between information spaces,
- r = the total number of resources.

Since, in the worst case, $a' = a$, the second pass is $O(a^2qr)$. Therefore, the Dynamic Approximation algorithm runs in polynomial time:

$$O(ar + aqr + a^2qr) \tag{9}$$

4.5.1.2 Even Splitter Algorithm

In addition to the Dynamic Approximation algorithm, we developed two additional inter-information space algorithms that serve two purposes. First, they are simple to implement, scalable, and likely to be sufficient for some situations. Second, they serve as baselines against which to evaluate the Dynamic Approximation algorithm, i.e., to see whether the additional processing required by the Dynamic Approximation algorithm provides significant improvement over the simple allocation of these algorithms and in what types of scenarios.

The *Even Splitter algorithm* provides coarse-grained allocations to information spaces in linear runtime. It takes the resources shared between multiple information spaces and divides them evenly between the information spaces, providing an equal amount of each shared resource to the information spaces sharing it. For example, if two information spaces share resource X, each would be allocated half of X.

4.5.1.3 Weighted Splitter Algorithm

The *Weighted Splitter algorithm* is also designed to provide coarse-grained allocation to information spaces in linear time. It takes the resources shared between multiple information spaces and allocates them based on a weighted factor. This factor can be provided by an administrator or derived from the information space priorities. For example, if two information spaces share a resource X, and one information space is twice as important as the other, the algorithm allocates two thirds of the resource to the more important information space and one third of the resources to the other information space.

4.5.2 Two-Phased QoS Management Algorithms Using the Inter-Information Space and Intra-Information Space Algorithms

The combination of first phase inter-information space choices and second phase intra-information space choices results in six total two-phase algorithms:

1. Dynamic Approximation + Greedy Approximation
2. Even Splitter + Greedy Approximation
3. Weighted Splitter + Greedy Approximation
4. Dynamic Approximation + Optimizing Brute Force
5. Even Splitter + Optimizing Brute Force
6. Weighted Splitter + Optimizing Brute Force

4.5.3 Coordination of the Two-Phase Algorithms

The QoS allocation algorithms can be run centralized or distributed. In a centralized QoS management system, there is no advantage to be gained in running the two-phase algorithm over running an intra-information space algorithm (i.e., Greedy Approximation or Optimizing Brute Force) over the global view of all applications in all information spaces, unless the centralized node has the ability to run the second phase algorithms for each information space in parallel. The centralized algorithm is simpler and takes care of some coordination (QoS allocations for all applications are available at the same time). However, the centralized algorithm will not scale as well and introduces a single point of failure.

In the distributed QMS, we run the second phase (the intra-information space algorithm) of the two-phase algorithm in parallel in each information space (sensible since the second phase algorithm only runs over the scope of an individual information space). The first phase can be run either distributed in each information space or in a centralized place. The former is possible because the algorithm is deterministic and will provide the same allocation of resources for all information spaces. The initial pass of the Dynamic Approximation algorithm requires knowledge of all applications in all information spaces.

In any case, the distributed or centralized approaches require synchronization and coordination to be carefully controlled. The algorithms produce allocations that consist of assigning QoS levels to applications, with any change of QoS levels relinquishing some resources and acquiring others. The delivery and enforcement of the allocation must be completed before any new reallocation is undertaken and resources must be relinquished before other applications can use them. We discuss our approach to synchronization and coordination in more detail in Section 6.

4.6 Summary Comparison of the Various QoS Management Algorithms

Table 2 presents a summary of the advantages and limitations of the algorithms that we developed under the DynRIIC project and presented in this section. Section 5 presents more detailed evaluations of the relative performance of the algorithms presented in this section.

Table 2: Summary comparison of the QoS management algorithms

Algorithm	Advantages	Limitations
Resource Allocation	• Fast • Requires only limited inputs about control points; simple input	• Doesn't consider the system dynamics and multi-resource interactions • Doesn't handle extreme resource contention (fixable with a starvation policy) • Based on a simple priority scheme (weights on system elements); does not optimize system utility
Optimizing Brute Force	• Provides an optimal solution • Finds a solution if one exists	• Not scalable to large scenarios • Highly variable runtime • Requires elaborate inputs, including control points and QoS levels
Greedy Approximation	• Scales to large scenarios (quadratic runtime) • High optimality on average	• Suboptimal solutions in some scenarios • Starvation of control points must be allowed • Requires elaborate inputs, including control points and QoS levels
MMKP	• Linear runtime	• Poor optimality for highly contentious scenarios • Significant tradeoff of speed and optimality
Two-Phased	• Scales better than centralized	• Performance best when information space size is nearly uniform and only a subset of applications share resources across information spaces • Currently requires centralized knowledge of applications to "discover" those that share resources

5 Evaluation of the QoS Management Algorithms

We conducted a set of experiments to evaluate the relative performance of the algorithms, in terms of quality of the solution produced and the speed of execution to reach a solution. This section describes the experiments that we conducted and their results. The full experimental process and results are described in a separate report [25].

The experiments showed the following significant results:

- The Optimizing Brute Force always produces an optimal allocation (i.e., the highest utility allocation possible within the available resources). While in the worst case its execution time is exponential in the number of applications, our heuristics provide good scalability up to moderate numbers of applications.
- The Greedy Approximation provides much better scalability and in many cases produces near optimal solutions. Some scenarios produce outliers, i.e., sub-optimal solutions. We determined that one cause of outliers is scenarios with extremely high resource contention that is unlikely to happen in practice.
- The MMKP algorithm exhibits good execution time performance and scalability with some choices for quantization, but poor optimality.
- The two-phase algorithms with the Dynamic Approximation first phase provides the best optimality in contentious and non-contentious scenarios. The two-phase algorithm produces solutions near the optimality of a centralized version of the algorithm but scales better. The two-phase algorithm executes the fastest when the applications are uniformly distributed among the information spaces and only a relatively small number of the applications share resources across the information spaces.
- The two-phase approximation algorithm (Dynamic Approximation-Greedy Approximation) provides more optimal solutions in highly contentious scenarios than the two-phase algorithms with the Even- or Weighted-Split first phase, because the first phase approximation algorithm considers contention for resources in its allocation, whereas the others do not.

5.1 Experimental Set up

5.1.1 Experiment Platform

We used a personal computer with a 2.80 GHz Intel® Pentium®-4 CPU with 512 KB RAM, running the Linux (Fedora Core Release 6) operating system.

5.1.2 Scenario Generator and Simulator

We developed a *scenario generator* that randomly generates scenarios that we used as input to a *simulator* that we developed to execute the algorithms on the scenarios. Each scenario consists of a set of applications, a set of QoS levels for each application, a utility value for each QoS level,

and a set of resources and amount used by each QoS level. The generator accepts the following arguments: the number of applications (control points) in the scenario, the number of QoS levels (service-levels) for each application, the total number of resources in an information space, and the number of resources (to be chosen from the total number of resources) for each QoS level. The generator produces a random value for utility for each combination of application and QoS level, randomly chooses the resources to use for each QoS level from among those available, and generates a random amount of each resource that is requested for each QoS level.

The simulator can simulate either a single information space or multiple information spaces. It takes as input a set of scenarios, runs the QMS algorithms, and produces the solution allocation, the utility of the solution, the runtime of the algorithm, and values for the metrics described in Section 5.2 as a comma separated file. The simulator simulates a single information space by taking scenarios generated by the generator and then running the intra-information space algorithms on each scenario. It simulates two information spaces by dividing the set of applications in a generated scenario evenly into two sets, each representing one of the information spaces. For each information space, the simulator runs the first-phase algorithms followed by the intra-information space second-phase algorithms.

5.1.3 Statistical Package and Presentation of Results

We used the R package [21] to analyze the data generated by the simulator, by passing the comma-separated files produced by the simulator as an input to the R package. To plot graphs we used the R package and rgl library [22] (a third-party extension to the R package). We plotted boxplots [3], lowess (regression line), and histograms using the R package. We plotted 3-D plots using the persp and persp3D functions of rgl.

Boxplots (www.wikipedia.org/Boxplots) display the interquartile *range (IQR)*, i.e., the range from the first quartile to the third in which the middle 50% of data values lie, as a box. A thick black line in the middle of the box represents the median. Vertical lines extending out from the box and ending in horizontal bars, called *whiskers*, represent the extent of the (non-outlier) observed values. Circles beyond the whiskers represent outliers, i.e., values above 1.5 × IQR + the upper quartile value or less than -1.5 × IQR below the lower quartile value.

5.1.4 Experimental Design

In general, for each of the experiments described in this report, we use the scenario generator to generate a sizable set of scenarios. For many of the intra-information space experiments, we generated scenarios with the following parameters: 3 QoS levels, 6 resources per QoS level, and 110 total resources. We varied the number of applications. For each application set, we generated 100 scenarios. For other experiments, we will describe the specific experiment design as we describe the experimental results.

5.2 Experimental Metrics

5.2.1 Algorithm Metrics

We collected the following metrics to compute the effectiveness (as a percentage of optimality) and the runtime of the QMS algorithms:

- *Percentage of Optimality:* The optimal solution is the feasible solution with the highest utility. For the solution returned by any algorithm, we compute its percentage of optimality by dividing its utility by the utility of the optimal solution[3]. For a given scenario, we use the optimality reported by the Optimizing Brute Force algorithm as the baseline against which the optimality of all the algorithms are compared.
- *Runtime:* Runtime is a measure of how fast an algorithm executes on a given hardware. We analyze worst case runtime and express it using *Big O* notation. We use the simulator to measure the runtime in our experiments. Although the absolute runtime depends on the hardware on which the algorithm is executed, the relative runtimes of various algorithms are comparable because we ran all our experiments on the same machine.

5.2.2 Contention Metrics

As part of our experiments, we evaluated the effect of *contention* on our algorithms, i.e., how resource rich or resource scarce the scenario is, and collected contention metrics to support this. Contention metrics measure the relation between the resources available and the resources that are requested in scenarios. We defined the following five contention metrics:

1. *Percent of infeasible solutions* measures the total number of infeasible solutions out of the total number of possible solutions (leaf nodes in the search tree created by the Optimizing Brute Force algorithm). For example, the total number of possible solutions (i.e., possible allocations) for 10 applications and 3 QoS levels is 59,049 solutions. If the Optimizing Brute Force algorithm finds only 200 solutions to be feasible, we compute the percent of infeasibility as (59049-200)/59049. The percent of infeasibility is directly proportional to the level of contention. That is, the higher the percentage of infeasible solutions, the higher the contention for resources in the scenario.
2. *Lowest percent of applications starved in any feasible solution* measures the smallest number of applications that are starved (i.e., that do not receive resource allocations) in any of the feasible solutions. This metric is directly proportional to the level of contention.

[3] Calculating the percentage of optimality requires knowing the optimal solution or, more specifically, the utility of the optimal solution. This requires running the Optimizing Brute Force algorithm to find the optimal solution, which limits the size and number of scenarios we can run.

3. *Highest percent of applications requesting the most-shared resource* measures the largest number of applications (out of the total number of applications) that request the same resource in a given scenario. This metric is directly proportional to the level of contention.
4. *Highest percent of resource requested by applications (in any QoS level) requesting the most-shared resource* measures the highest amount of a resource requested in any possible (not necessarily feasible) allocation. For example, if a scenario has an allocation in which 500% (i.e., 5x of what is available) of a resource is requested and another scenario has no more than 95% of any resource requested, the first scenario indicates a more severe potential bottleneck than the second. This metric is directly proportional to the level of contention.
5. *Percentage of resources shared across the information spaces* affects the size of the space over which the first phase algorithms operate. It also indicates a measure of contention for those resources shared between information spaces.

5.2.3 Distribution of Scenarios

Because metrics 3-5 relate specifically to attributes of scenarios, they are affected by our scenario generator more than specifically by attributes of our algorithms. In our experiments, we generated large numbers (typically 50,000) of scenarios randomly, with the intent to generate scenarios with a wide range of values for these metrics. In order to evaluate how well the scenarios we generated covered the space, we calculated metrics about the distribution of the randomly generated scenarios. Specifically, we examined the statistical distribution of the scenarios with respect to our contention metrics. As presented in more detail in [25], we expected and observed a *normal* distribution, in which there were more scenarios exhibiting medium amounts of contention than exhibiting extremely high or extremely low amounts of contention. We also calculated the distribution of scenarios to ensure that we had a sufficient number of data sets for the different values of our contention metrics against which we were comparing, thereby increasing our confidence in the results we observed. More details are presented in [25].

5.3 Percent of Optimality and Runtime of the Intra-Information Space Algorithms

5.3.1 Optimizing Brute Force

The Optimizing Brute Force algorithm always produces an optimal solution (i.e., 100% optimality). Hence, we use this as the baseline algorithm for measuring the effectiveness of the other algorithms.

However, in the worst case Optimizing Brute Force runs in exponential time. Furthermore, the runtime grows exponentially as either the number of applications or the number of QoS levels within the applications increase. Figure 18 shows boxplots of the results for an experiment in which we generated scenarios with the number of applications varying from 10 to 110 by steps

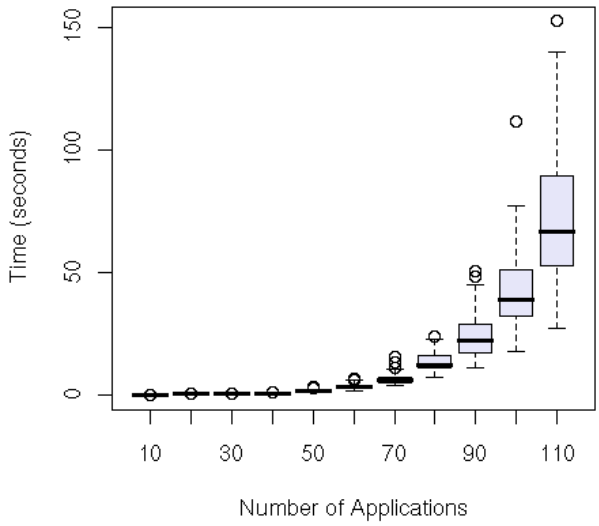


Figure 18: Impact of varying number of applications on the runtime of the Optimizing Brute Force algorithm.

of 10, with 100 scenarios at each step (see Section 5.1.3 for an explanation of boxplots). Each application had 3 QoS levels, and each QoS level used 6 resources selected randomly from a total of 110 scenarios. The runtime is good (near one second) until about 40-50 applications, after which the median runtime and the variance in runtime increase dramatically. The median runtime increases to about 70 seconds at 110 applications, with a worst case runtime of 150 seconds. The increased variance is due to the difference in pruning possible from scenario to scenario. The scenarios with the highest runtime must result in little pruning, causing the Optimizing Brute Force algorithm to search nearly the entire space. In contrast, the best measured runtime (about 25 seconds for 110 applications – 6x faster than the worst case time) must be with scenarios that allow significant pruning (i.e., many infeasible solutions and/or quickly found high-utility solutions).

Figure 19 depicts the increase in the runtime of Optimizing Brute Force when either the number of QoS levels or the number of applications increases. For this experiment, we generated scenarios that varied the number of QoS levels from 1 to 20 for each number of applications and that varied the number of applications from 1 to 20 for each number of QoS levels. The runtime is acceptable up to about 10 of either, then increases dramatically.

5.3.2 Greedy Approximation

Our experiments indicate that the Greedy Approximation algorithm produces solutions that are close to optimal, with a significant improvement in runtime over the Optimizing Brute Force baseline. The boxplot in Figure 20 represents an experiment in which we ran the Greedy

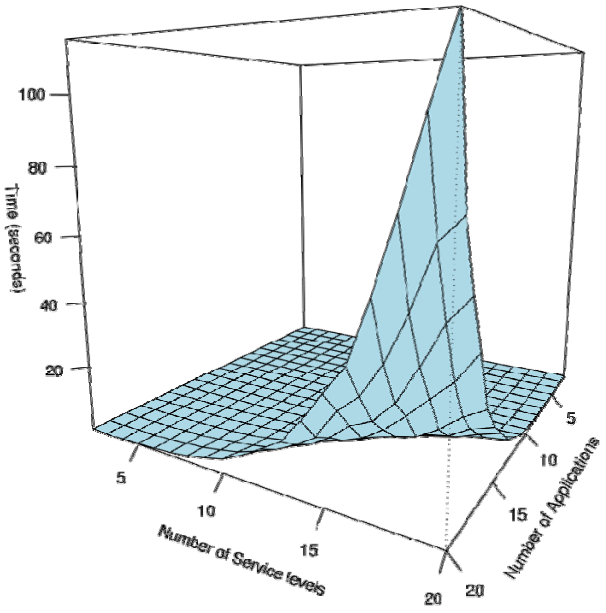


Figure 19: Impact of simultaneously varying number of applications for a given QoS level and number of QoS levels for a given application when running the Optimizing Brute Force.

Approximation algorithm on 50,000 scenarios, with 10 applications[4], 3 QoS levels for each application, 3 resources per QoS level, and 30, 70, 110, 150, and 190 total resources (10,000 scenarios for each level of total resources). We found that the median solution is 96% of optimal, and 75% of the solutions are over 90% of optimal or better, with all but the outliers producing solutions 80% optimal or better. The worst solution is 40% of optimal.

Our experiments also indicate that contention adversely impacts the effectiveness of Greedy Approximation. Specifically, we observed the median optimality decline to 75% as the level of contention increases significantly. Figure 21 illustrates experiments run with the number of applications varying from 10 to 110, 3 QoS levels, 6 resources per QoS level, and 20 total resources. In these experiments, the median percentage of optimality varied between only 75-85%, although the worst case percentage optimality is approximately the same as the experiment in Figure 20.

The difference in the number of resources being used by each application and the number of resources available causes the experiments depicted in Figure 20 and Figure 21 to exhibit different contention characteristics. The experiments depicted in Figure 20 (selecting 3 resources from 30, 70, 110, 150, or 190 resources) had scenarios with the percentage of feasible solutions ranging from under 10% to 100%, whereas in the experiments depicted in Figure 21 (selecting 6 resources from 20 available), all of the scenarios had fewer than 0.5% feasible allocations. This

[4] We had to generate scenarios with a modest number of applications in order to have an optimality baseline against which to compare, since we have to run the Optimizing Brute Force algorithm on each of the 50,000 scenarios to get the optimal solution.

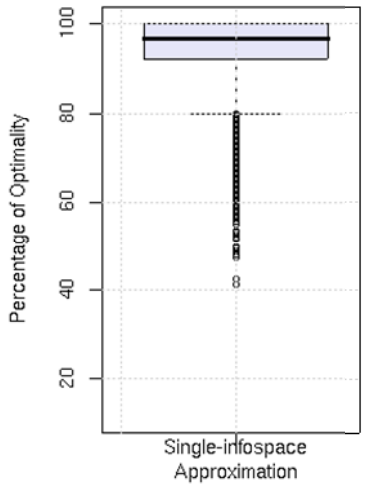


Figure 20: Optimality of the Greedy Approximation algorithm on 50,000 scenarios with 10 applications, 3 QoS levels per application, 3 resources per QoS level, and 30, 70, 110, 150, and 190 total resources (10,000 scenarios each).

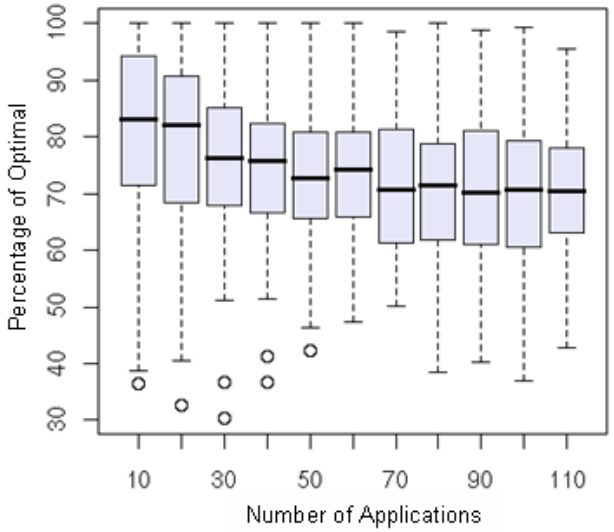


Figure 21: Optimality of Greedy Approximation on 50,000 scenarios with a varying number of applications, 3 QoS levels per application, 6 resources per QoS level, and 20 total resources.

provides strong evidence that the level of contention affects the optimality of the Greedy Approximation algorithm.

Effectiveness of the initial penalty optimizing factor. As described in Section 4.3, we enhanced the base Greedy Approximation algorithm with an initial penalty vector. We introduced the initial penalty vector to handle a set of scenarios (that we dubbed *Greedy Achilles' Heel* scenarios) that produced sub-optimal solutions in the base algorithm. These scenarios have one or more high utility applications that request a significant amount of a highly contended resource. Since the base algorithm treated all resources equally and completely available at the

beginning, these applications would be greedily assigned resources and potentially starve a large number of other applications resulting in a significantly suboptimal solution. To prevent this, we enhanced the algorithm to perform an initial pass and assign an initial penalty to highly contended resources, making it cost more to request these resources.

We conducted experiments to evaluate the effectiveness of the initial penalty enhancement. For this experiment, we generated Greedy Achilles' Heel scenarios with a varying number of applications, 3 QoS levels for each application, and 6 resources selected randomly from 110 resources for each QoS level. We varied the number of applications from 10 to 40 in steps of 10 (again, the upper bound of 40 is so we could run the Brute Force algorithm to get the optimal solution against which to compare). For each number of applications, we had 100 scenarios on which we ran the Greedy Approximation algorithm both with and without the initial penalty. Our results, illustrated in Figure 22, show that the initial penalty improves the percent of optimality significantly for this class of scenarios. Without the initial penalty, Greedy Approximation provides a low median percent of optimality ranging from approximately 30% to approximately 42% (Figure 22a). When we add the initial penalty to Greedy Approximation, the median percent of optimality on the same set of scenarios improved to a range of 75% to 85% (Figure 22b). Notice that the percent of optimality declines as the number of applications increase in both cases, due to an increase in contention (more applications competing for the same number of resources).

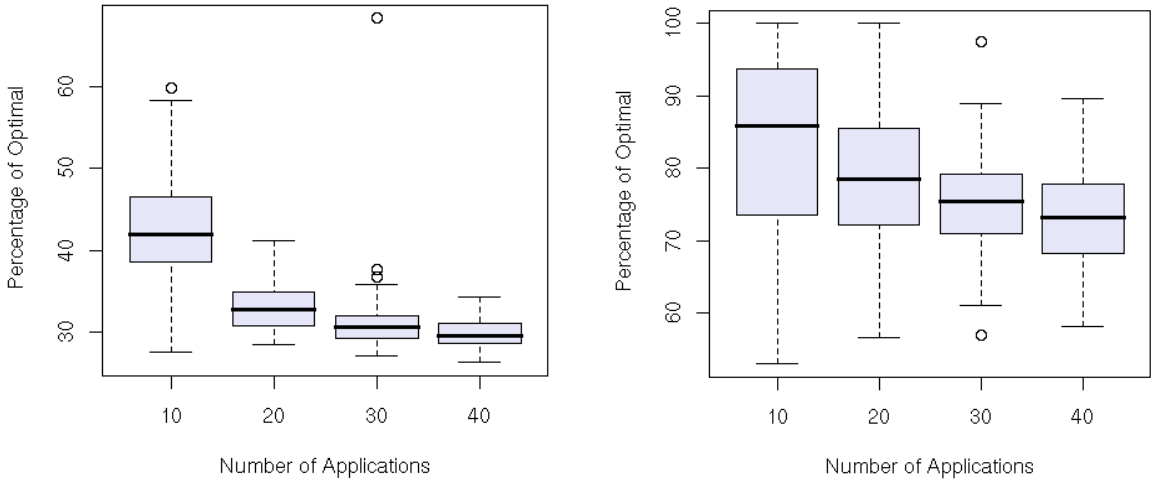


Figure 22: Percentage of optimality of Greedy Approximation for Greedy Achilles' Heel scenarios (a) without the initial penalty optimization (b) with the initial penalty optimization.

Runtime performance of Greedy Approximation. We evaluated the runtime of Greedy Approximation in order to examine the feasibility of using Greedy Approximation in information spaces with large numbers of applications. We analyzed the worst-case runtime and conducted experimental evaluations to measure the runtime.

Pseudocode for the Greedy Approximation algorithm follows:

```
1:  initializeList(cp+sl-List)
2:  while(cp+sl-List not empty) {
3:    next = find_max_utility(cp+sl-List);
4:    addToUsedResources(next.resourceUsage)
5:    removeChosenCP's Other Service Levels
6:    removeInfeasible(cp+sl-List)
7:  }
```

Step 1 is the creation of the initial penalty vector, and takes a single pass through $q*a$ elements, where q is the number of QoS levels and a is the number of applications. The loop bounded by step 2 and 7 is executed at most a times, since step 5 removes at least q elements from the list each time through (the list begins with $q*a$ elements). Step 6 could remove more, so the actual number of times through the loop could be fewer than a times. Steps 3 and 4 are linear time operations on the current list of applications × QoS levels and resources, respectively.

Therefore, the worst case runtime is equal to $(aq) + a(arq)$, or $O(a^2qr + aq)$, where:

- a is the number of applications,
- q is the number of service levels, and
- r is the number of resources

Furthermore, notice that the operation in step 6 affects the runtime of future iterations. If step 6 prunes a significant number of infeasible allocations from the *cp+sl-List*, then the number of times through the loop at step 2 is significantly reduced. In scenarios where 100% of solutions are feasible, step 6 will never remove anything and the algorithm will run in worst case time. In scenarios where step 6 removes most of the elements because many allocations are infeasible, the algorithm will run much faster.

From this analysis, Greedy Approximation's best runtime happens in scenarios with extremely high contention (step 6 does a lot), and its worst-case runtime is in scenarios with low contention (step 6 does nothing). Since we showed above that Greedy Approximation produces better solutions (in terms of percentage of optimality) when contention is low, this means that Greedy Approximation is fastest when it does the worst job and is slowest in those scenarios in which it comes up with near optimal solutions.

Regardless, in worst case its runtime is polynomial or, more precisely, *quadratic* in the number of applications.

We also evaluated the runtime by running experiments that varied the number of applications and the number of resources, the two scenario attributes that we believed might scale to large numbers in realistic scenarios.

For the experiment with varying number of applications, we increased the applications from 10 to 300 in steps of 10, with 100 scenarios for each discrete number of applications. Each application had 3 QoS levels, and each QoS level used 6 resources selected randomly from a total of 110 resources. As expected from the analysis above, we observed that the runtime of Greedy Approximation increases polynomially with the increase in the number of applications (Figure 23). As comparison, executing the Greedy Approximation algorithm on a randomly generated scenario with 110 apps took less than 0.10 seconds versus 60 seconds for the

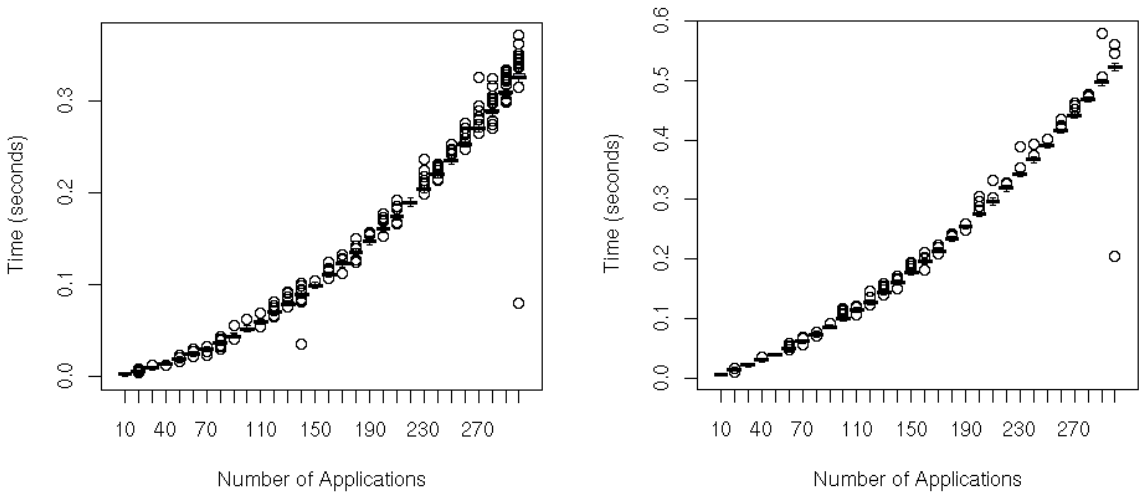


Figure 23: Runtime of the Greedy Approximation algorithm as the number of applications increase. (a) For the Greedy Approximation algorithm without initial penalty. (b) Greedy Approximation algorithm with initial penalty.

Optimizing Brute Force algorithm. Figure 23 also shows that the optimization to add the initial penalty, which greatly improves the optimality of the algorithm for some scenarios, impacts the runtime by a constant multiplicative factor, due to the initial linear pass through the application×QoS level list. We observed subsecond runtimes for up to hundreds of applications (0.6 seconds for 300 applications, 0.3 seconds for 300 applications without the initial penalty optimization).

Figure 24 illustrates the results of an experiment to evaluate the effects of varying the number of resources. In this experiment, we randomly generated scenarios with 100 applications, 3 QoS levels per application, 3 resources per QoS level, and total resources varying from 30 to 180, in steps of 10. We had 100 scenarios for each discrete number of resources. Our results indicate that the runtime of Greedy Approximation grows approximately linearly as the number of resources increases, as shown in Figure 24, confirming what we expected from our analysis above.

5.3.3 MMKP

To evaluate the percent of optimality of the MMKP algorithm, we used the experiment described in Section 5.1.4. We generated scenarios with the number of applications varying from 10 to 110 by steps of 10, with 100 scenarios at each step. Each application had 3 QoS levels, and each QoS level used 6 resources selected randomly from a total of 110 scenarios.

Our MMKP algorithm normalized and approximated the total resources used by an application from 0 to 1 and iterated over them by a step of 0.1. This results in resources being allocated in tenths of their total amount available. This results in fast scalable runtime for the algorithm. As shown in Figure 25b, the runtime for MMKP is linear and scales well, with 300 applications taking about 0.3 seconds.

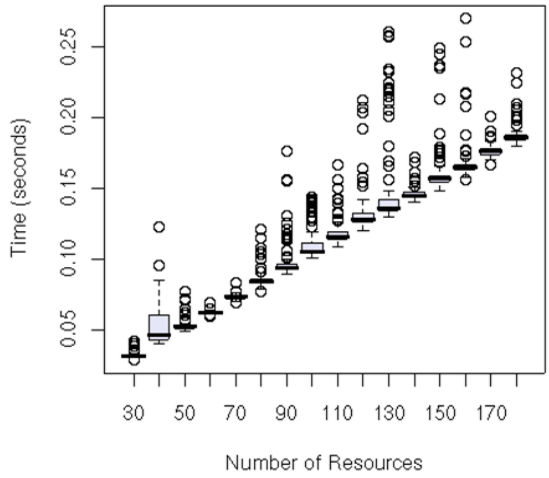


Figure 24: Runtime of the Greedy Approximation algorithm as the number of resources increases.

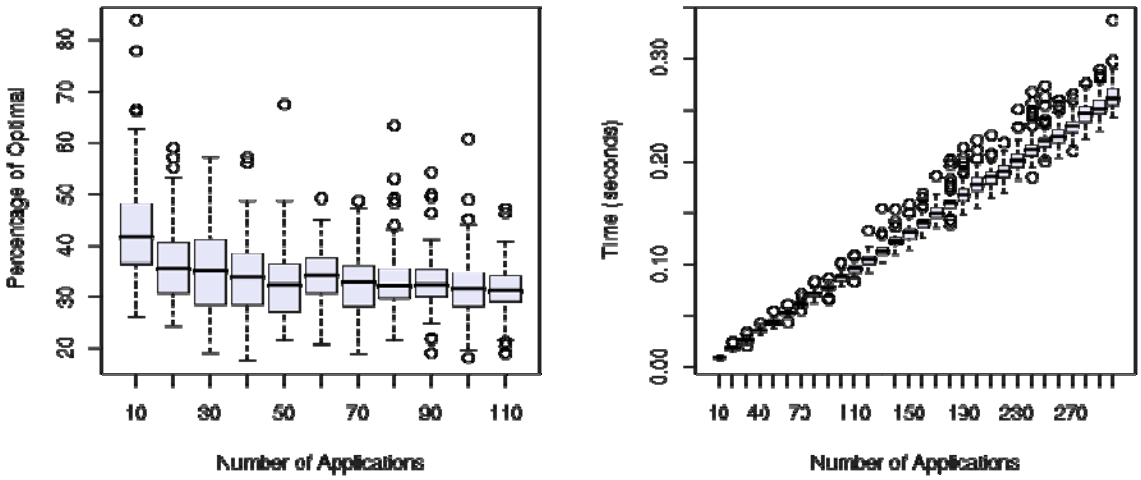


Figure 25: (a) Percent of optimality of the MMKP algorithm. (b) Runtime of the MMKP algorithm.

However, the results of the experiments also indicate that MMKP (with 0.1 quantization) provides a median of only 35% optimality. As described in Section 4.4, dynamic programming, upon which MMKP is based, is pseudo-polynomial (i.e., polynomial in the value of its input). A strict MMKP dynamic programming algorithm might use the actual value of the amount of resources (instead of normalizing to a scale of 0-1) and a step of 1 (or whatever units the resources could be allocated in) for all the resources, but in doing so would execute closer to exponential time than polynomial time. The choice of a quantization is necessary to make the algorithm behave better than an exponential time algorithm. While 0.1 quantization is very coarse grain (e.g., if an application and QoS level requests 3% of a resource, it will get either 0% or 10%), we expect a finer grained quantization to significantly impace execution time performance.

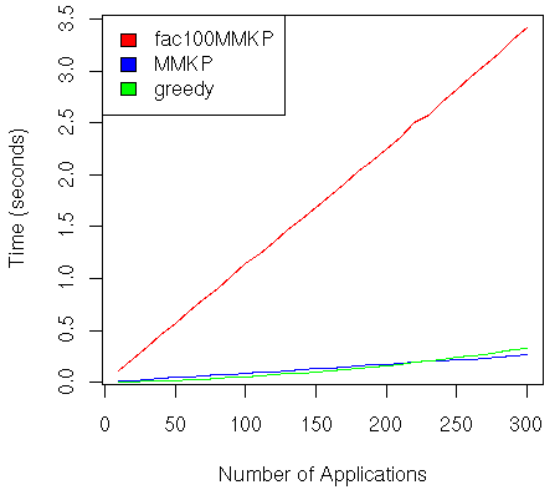


Figure 26: Comparison of the runtime of MMKP with 0.01 quantization, MMKP with 0.1 quantization, and the Greedy Approximation algorithm.

Figure 26 shows a comparison of MMKP with 0.01 quantization (fac100MMKP), MMKP with 0.1 quantization (MMKP) and the Greedy Approximation algorithm (greedy). As expected, the runtime for MMKP with 0.01 quantization is approximately 10x that of the MMKP with 0.1 quantization (over 3 seconds versus 0.3 seconds for 300 applications). At these numbers of applications, the execution time for MMKP with 0.1 quantization is similar to that of the Greedy Approximation. Because of this and the significantly better optimality of the Greedy Approximation algorithm, we did not continue developing or evaluating the MMKP algorithm.

5.4 Percent of Optimality and Runtime of the Two-Phased Algorithms

In these experiments, we analyzed the percent of optimality and runtime of the two-phased algorithms, which each run one of the inter-information space algorithms (i.e., Dynamic Approximation, Even Splitter, or Weighted Splitter) in the first phase and an intra-information space algorithm (i.e., Greedy Approximation or Optimizing Brute Force) as the second phase. All the experiments described from here on use the experimental design described in the following section, unless stated otherwise.

5.4.1 Experimental Design

We designed experiments to run the algorithms on a large enough number of randomly generated scenarios to ensure that we would have a significant number of scenarios exhibiting various characteristics (e.g., a varying amount of resource contention, a varying percentage of feasible allocations, and a varying number of resources shared between information spaces).

The three inter-information space algorithm choices for the first phase and the two intra-information space algorithm choices for the second choice result in six total two-phased algorithms as the experimental cases:

1. Even Splitter-Optimizing Brute Force
2. Weighted Splitter- Optimizing Brute Force
3. Dynamic Approximation- Optimizing Brute Force
4. Even Splitter-Greedy Approximation
5. Weighted Splitter-Greedy Approximation
6. Dynamic Approximation-Greedy Approximation

As two baselines against which to compare, we used "centralized" versions of both the Optimizing Brute Force and the Greedy Approximation algorithms. The centralized Optimizing Brute Force provides the optimal allocation against which to compare, while the centralized Greedy Approximation provides a comparison measure of the percentage optimality and speed of a centralized algorithm.

Our experiments consisted of executing the algorithms on 10,000 randomly generated *scenarios*. Each scenario consists of ten applications, each with three service levels (high, low, and starvation), with each service level utilizing three resources.[5] The scenario generator varies the resources used by each service level, the amount of each resource used by each service level, and the utility associated with each service level.

We conducted the experiments using a simulator that takes each scenario and divides the applications between two information spaces, putting half (i.e., five) in each. It then runs the first-phase algorithms to divide the shared resources followed by the second-phase algorithms for each information space to choose an allocation. It also runs the two centralized algorithms on each information space to gather the baseline metrics.

The number of applications sharing available resources and the percentage of resources shared across information spaces varied from scenario to scenario, but are affected not only by the factors that the scenario generator varies (i.e., resources used by each service level and the amount of each resource used by each service level) but also by the total number of resources available. To attempt to get good representative coverage over a variety of possible scenario configurations, we generated 10,000 scenarios each for a varying number of total available resources: 30, 70, 110, 150, and 190 resources. This results in a total of 50,000 scenarios.

5.4.2 Percent of Optimality of the Two-Phased Algorithms

Figure 27a illustrates the results for the experiment described above for the two-phase algorithms with Optimizing Brute Force as the second phase (algorithms 1-3). Figure 27b illustrates the results for the experiment described above for the two-phase algorithms with Greedy Approximation as the second phase (algorithms 4-6). Our results demonstrate that Dynamic-Approximation as the first-phase provides the best percent of optimality with either Optimizing Brute Force or Greedy Approximation as the second phase. Specifically, Dynamic Approximation provides a median of 100% of optimality when the Optimizing Brute Force

[5] We chose the number of applications (a) and service levels (q) to support running a large number of scenarios against the centralized Optimizing Brute Force algorithm baseline, which takes $O(q^a)$ to run. Ten applications with three service levels results in a search space of 59,049 possible allocations. Twenty applications with the same number of service levels drive the search space up to over 3 billion possible allocations.

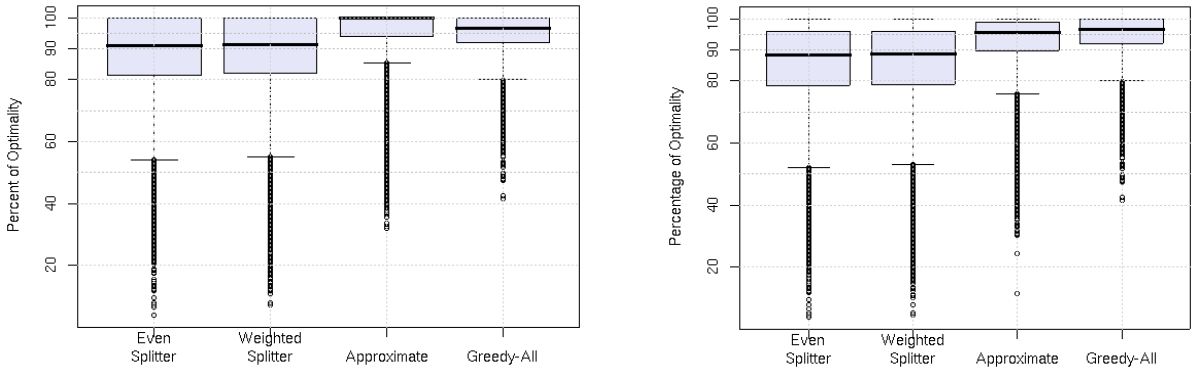


Figure 27: Comparison of percent of optimality of the two-phase algorithms with (a) Optimizing Brute Force as the second phase and (b) Greedy Approximation as the second phase. (Approximate refers to the Dynamic Approximation first phase.)

algorithm is used as the second-phase algorithm (Figure 27a) and a median of 96% of optimality when Greedy Approximation is used as the second phase (Figure 27b). The Even Splitter and Weighted Splitter algorithms provide medians of 92% percent of optimality when Optimizing Brute Force is used as their second phase (Figure 27a) and nearly 90% when Greedy Approximation is used as the second phase. Moreover, the percent of optimality of the two-phase algorithm with Dynamic Approximation as the first-phase with either of the second phases is nearly equivalent to the percent of optimality of the centralized approximation algorithm, Greedy-All (running the Greedy Approximation algorithm with all the applications in all the information spaces).

5.4.3 Runtime of the Two-Phased Approximation Algorithm

This section presents an evaluation of the runtime of the *Dynamic Approximation-Greedy Approximation* algorithm, which is likely to be the best choice for many scenarios because (a) its first phase outperforms the other choices and (b) its second phase runs in quadratic time versus the exponential time of the alternative. First, we analyze the worst case runtime of the algorithm. Then, we show the results of experimental evaluations of the runtime in which we vary both the number of applications and number of resources.

Analysis of the runtime. In Section 4.5.1.1, we determined that the Dynamic Approximation first phase takes $ar + aqr + a'^2qr$, where a is the number of applications, q is the number of QoS levels, r is the number of resources, and $a' \subseteq a$ is the subset of applications that share resources between the information spaces. The Greedy Approximation second phase takes a_{max}^2qr time, where $a_{max} = max(a_1, ..., a_I)$ and a_i is the number of applications in information space i, for $i=1..I$. That is, the second phase takes as long as the largest information space, i.e., the information space with the largest number of applications.

In a representative case in which, say, 10% of the total applications share resources across information spaces and the applications are evenly divided between the information spaces, the runtime would be $ar + aqr + (.1a)^2qr + (a/I)^2qr$. Which term dominates depends on which is larger, the number of applications sharing resources across the information spaces or the number of applications in each information space.

In the worst case, all of the applications would share resources across information spaces ($a' = a$) and applications would not be evenly spread across the information spaces (one information space would have $a-(I-1)$ applications and each of the rest of the information spaces would have one application each). In this case, the first phase essentially runs a centralized Greedy Approximation algorithm and the second phase is superfluous. The runtime in this worst case is as follows:

$$O(ar + aqr + a^2qr + (a-I+1)^2qr) = O(ar + aqr + 2a^2qr) = O(ar + aqr + a^2qr) \quad (10)$$

Experimental evaluation of runtime. Even though we just showed the centralized Greedy Approximation algorithm should perform better than Dynamic Approximation-Greedy Approximation in the worst case, we expect in the general case that the Dynamic Approximation-Greedy Approximation algorithm should run faster than Greedy-All. This is because we expect in the usual case a' should be less than a and that applications should be well distributed among information spaces.

We ran experiments to test the runtime of the two-phase algorithm as the following variables vary:

1. The number of total applications (and therefore the number of applications in each COI)
2. The number of resources

5.4.3.1 The Effect on Runtime of Varying the Number of Total Applications

Our analysis above showed that the number of total applications should significantly impact the runtime of the centralized and two-phase algorithms. However, we hypothesize that in the usual case where only a subset of applications share resources across information spaces and the total number of applications are well distributed between information spaces, that an increase in the total number of applications should impact the centralized Greedy-All algorithm more significantly than the two phase Dynamic Approximation-Greedy Approximation algorithm. In other words, we expect the two-phase approximation algorithm to scale better than the centralized version.

We tested our hypothesis by conducting an experiment in which we varied the total number of applications. We generated scenarios that varied the total number of applications from 20 to 2000, with 100 random scenarios for each data point. We forced an even distribution of applications between two information spaces (I=2) by dividing the applications in half, with each information space having half of the total number of applications (i.e., 10 to 1000 applications each). We set the scenario generator parameters to 3 QoS levels per application, 3 resources used per QoS level, and 110 total number of resources. The scenario generator randomly generated the utility value for each QoS level, the specific resources (from the 110 available) used by each QoS level, and the amount of each resources used. The choice of 110 available resources (a fairly large number compared to the number of resources, 3, used by each application) makes it more likely that the number of applications sharing resources across the information spaces is a *subset* of the total number of applications (i.e., the more available resources to choose from, the less

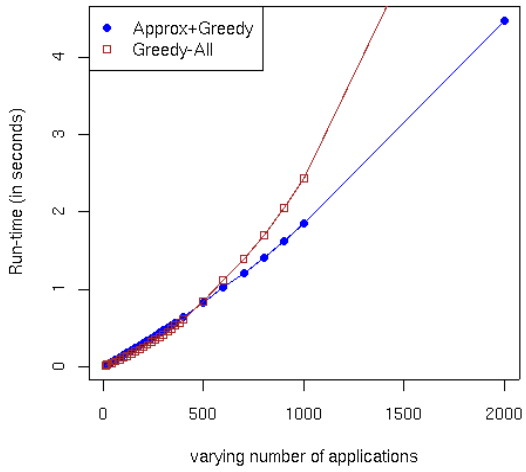


Figure 28: Median runtime of Dynamic Approximation-Greedy Approximation (Approx+Greedy) when we varied the number of applications from 20 to 2000. The centralized approximation algorithm, Greedy-All, is shown as a baseline.

likely that two applications will share resources). We plotted the median runtime for running the algorithms on each set of scenarios.

The results (Figure 28) indicate that as the total number of applications increases, the median runtime of both the Dynamic Approximation-Greedy Approximation algorithm (shown with blue line and circles) and the Greedy-All algorithm (shown with red line and squares) increase. The centralized Greedy-All algorithm outperforms the two-phase algorithm slightly at low numbers of total applications (up to a few hundred per information space). As the number of applications increase, the runtime of the two-phase algorithm increases at a much lower slope than the centralized algorithm, indicating that the two-phase algorithm scales much better in terms of number of total applications. Note that this experiment was conducted with two information spaces. The difference between the slopes would be even larger with more information spaces.

5.4.3.2 The Effect on Runtime of Varying the Number of Total Resources

Because of the linear contribution of the *r* term to equation (10), we hypothesize that the runtime of Dynamic Approximation-Greedy Approximation should change linearly as the total number of resources changes.

To test this hypothesis, we conducted an experiment in which we varied the total number of resources. As in the experiments above, we used the scenario generator simulation software to generate random scenarios varying the total number of resources from 10 to 150 in steps of 10. We set the scenario generator parameters to 50 applications, 3 QoS levels per application, and 3 resources used per QoS level. We divided the applications into two information spaces (I=2) such that each information space had half of the total number of applications (i.e., 25 applications). We used 5000 scenarios for each experimental data set and plotted the results as boxplots. The results in Figure 29 indicate an approximate linear slope for the median (thick dark line) runtime as the number of resources increases.

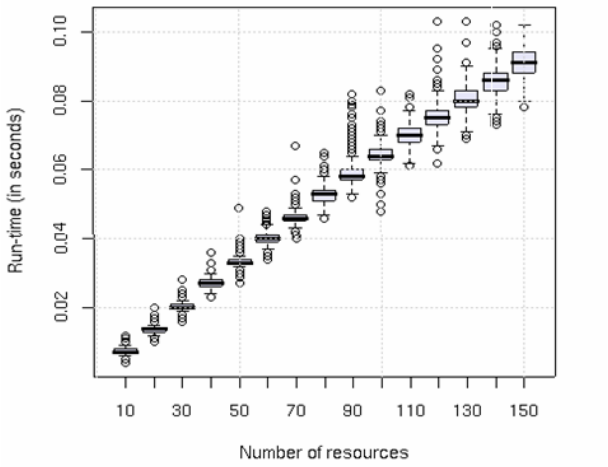


Figure 29: Runtime of the Dynamic Approximation-Greedy Approximation algorithm as a function of varying the number of resources.

5.4.4 Percentage of Optimality under Varying Levels of Contention for the Two-Phased Algorithms

Our earlier experimental results on Greedy Approximation indicated that the level of contention adversely impacts its percent of optimality. This motivated us to conduct experiments to examine how changes in the level of contention impact the percent of optimality and runtime of the two-phased algorithms.

For evaluating the percentage of optimality, we used the experiment described in Section 5.4.1, which utilized 50,000 randomly generated scenarios. We extracted the contention metrics defined in Section 5.2.2 from the scenarios, allowing us to identify subsets of the scenarios exhibiting various levels of contention. Plotting the percentage of optimality for these subsets allowed us to identify how the contention metric affects the optimality of the algorithms. In the following section, we present and discuss these results.

5.4.4.1 Contention Metric 1 – Percent of Infeasible Solutions

An allocation is considered infeasible if the amount of resource requested (for the applications and QoS levels in the allocation) exceeds the amount available for any of the requested resources. Therefore, a high percentage of infeasible allocations implies a high level of contention for resources. Figure 30 illustrates a regression graph showing the trend of the percentage of optimality as the percentage of infeasible solutions increases (and therefore contention increases). The Dynamic Approximation first phase provides the best percent of optimality with either Optimizing Brute Force or Greedy Approximation as the second phase. Moreover, Dynamic Approximation performs better in contentious environments when contrasted with Even Splitter to Weighted Splitter algorithms. As the percentage of infeasible solutions increase, the *Dynamic Approximation-Optimizing Brute Force* and *Dynamic Approximation-Greedy Approximation* algorithms eventually start to decline in percentage of optimality, but less than the other two-phase algorithms. Furthermore, they maintain a trend of 90% optimality or better and perform nearly as well as the centralized Greedy-All.

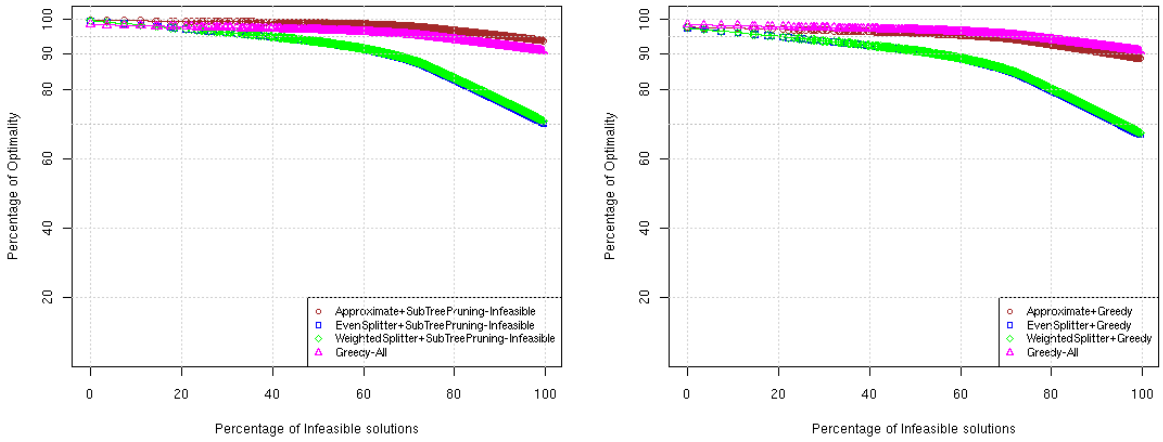


Figure 30: Using the "percent of infeasible solutions" metric for comparing percent of optimality of various inter-information space first phases with (a) Optimizing Brute Force as the second phase, and (b) Greedy Approximation as the second phase.

5.4.4.2 Contention Metric 2 – Lowest Percent of Applications Starved in any Feasible Solution

When there is significant contention for resources, it is possible that only some of the applications requesting the contended resources can be run. All others are *starved*, i.e., are allocated no resources and therefore cannot run. This metric looks at the feasible allocations and determines the lowest percentage of applications starved in any of them.

Figure 31 illustrates the regression graph showing the trend of the percentage of optimality as the lowest percentage of starved applications increases (and therefore contention increases). As

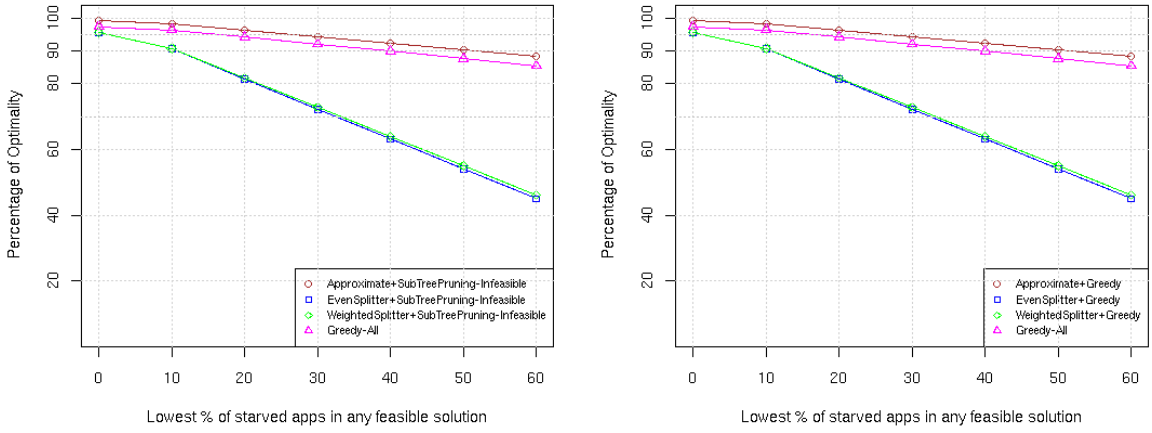


Figure 31: Using the "lowest percent of applications starved in any feasible solution" metric for comparing percent of optimality of various inter-information space first-phases with (a) Optimizing Brute Force as the second phase, and (b) Greedy Approximation as the second phase.

the percentage of starved applications increases, the trend of all the algorithms is to decrease in percentage of optimality. However, the two-phase algorithms with the Dynamic Approximation first phase decrease much more gradually and track the centralized algorithm well. Furthermore, they perform significantly better than the other two-phase algorithms when contention is higher, maintaining a trend of approximately 85% of optimal or better even when 60% or more of the applications are starved. In contrast, the two-phase algorithms with the Even Splitter and Weighted Splitter first phase trend downward to approximately 40% of optimal as the percentage of starved applications increases.

5.4.4.3 Contention Metric 3 – Highest Percent of Applications Requesting the Most-Shared[6] Resource

This metric looks at the largest number of applications requesting a single resource. All of the algorithms decline in performance as the percentage of applications requesting the most shared resource increases. Figure 32 illustrates the regression graph showing the trend of the percentage of optimality as the highest percentage of applications requesting the most shared resource increases (and therefore contention increases). As the percentage of applications requesting the most-shared resource increases, the trend of all the algorithms is to decrease in percentage of optimality. However, the two-phase algorithms with the Dynamic Approximation first phase decrease much more gradually and track the centralized algorithm well. Furthermore, they perform significantly better than the other two-phase algorithms when contention is higher, maintaining a trend of approximately 85% of optimal or better even when as many as 70% of the applications request the most-shared resource. In contrast, the two-phase algorithms with the Even Splitter and Weighted Splitter first phase trend downward to approximately 60% of optimal as the percentage of applications requesting the most-shared resource increases.

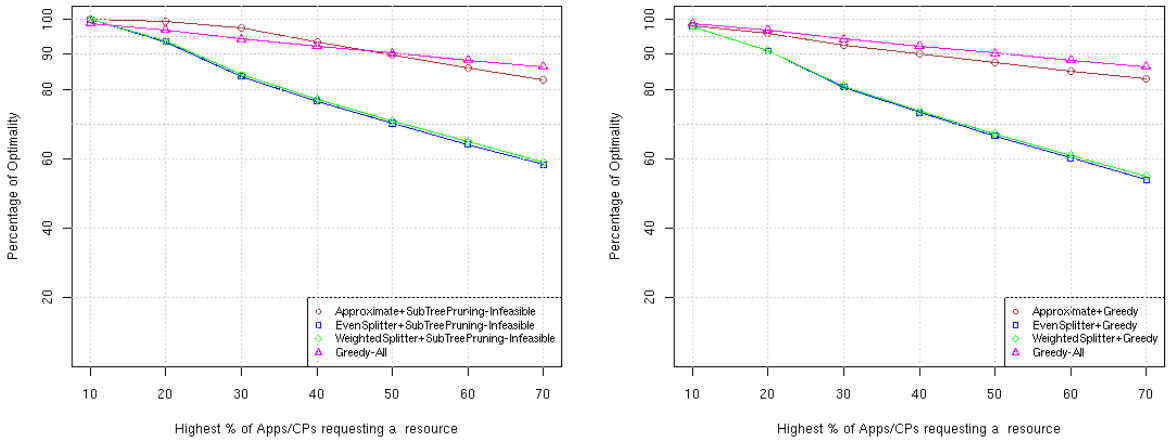


Figure 32: Using "highest percent of applications requesting a resource" metric for comparing percent of optimality of various inter-information space first-phases with (a) Optimizing Brute Force as the second phase, and (b) Greedy Approximation as the second phase.

5.4.4.4 Contention Metric 4 – Highest Percent of Resource Requested by Applications

While Figure 32 illustrates the effects of many applications requesting a specific resource, it does not take into account how much of the resource they collectively request. Therefore, this metric looks at the highest amount of a resource requested in any possible allocation. All of the algorithms decline in percentage of optimality as the highest percentage of resource requested by applications requesting the most-shared resource increases.

Figure 33 illustrates the regression graph showing the trend of the effect of the amount of a single, most requested (by quantity requested) resource on the optimality of the algorithms. The *x*-axis in the graphs represent the amount requested of the most requested resource, varying from less than half of the resource capacity requested (0.5x) to more than 5x (i.e., 500%) of the resource capacity. As would be expected, when less than the resource capacity (< 1.0) is the most requested, i.e., the scenario represents a resource rich environment, all algorithms trend toward optimal or near optimal solutions. As the amount requested increases, the optimality declines, although the algorithms with the approximation first phase still trend toward near optimal results, a gradual decline, and results close to the centralized algorithm, with a trend towards about 90 percent optimality.[7]

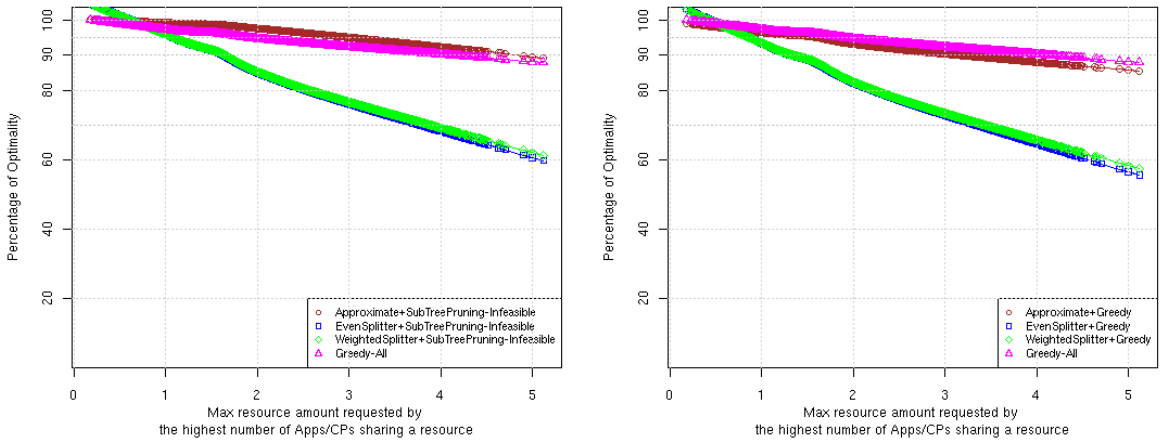


Figure 33: Using "highest percent of resource requested by applications (in any QoS level) requesting the most-shared resource" metric for comparing percent of optimality of various inter-information space first-phases with (a) Optimizing Brute Force as the second phase, and (b) Greedy Approximation as the second phase.

---

[6] The resource shared by the most number of applications.

[7] R's *lowess* function computes regression graphs that smooth out the scattered plots of the actual values. Since the slope of the Even Splitter and Weighted Splitter plots are so steep, it smoothes out to graphs whose leftmost points trend above 100% optimality in Figure 33, even though none of the actual values are.

5.4.4.5 Contention Metric 5 – Percentage of Resources Shared Across the Information Spaces

Figure 34 illustrates the regression graphs showing the trend in percentage optimality as the percentage of resources shared between information spaces increases. The two algorithms that use the Dynamic Approximation first phase perform very well, trending toward a gradual decrease in percentage optimality, contrasted with the more drastic drop in the other two-phase algorithms. *Dynamic Approximation-Optimizing Brute Force* and *Dynamic Approximation-Greedy Approximation* perform close to the centralized algorithm and indicate a trend toward approximately 90% optimality as the percentage of shared resources increases to a significant amount.

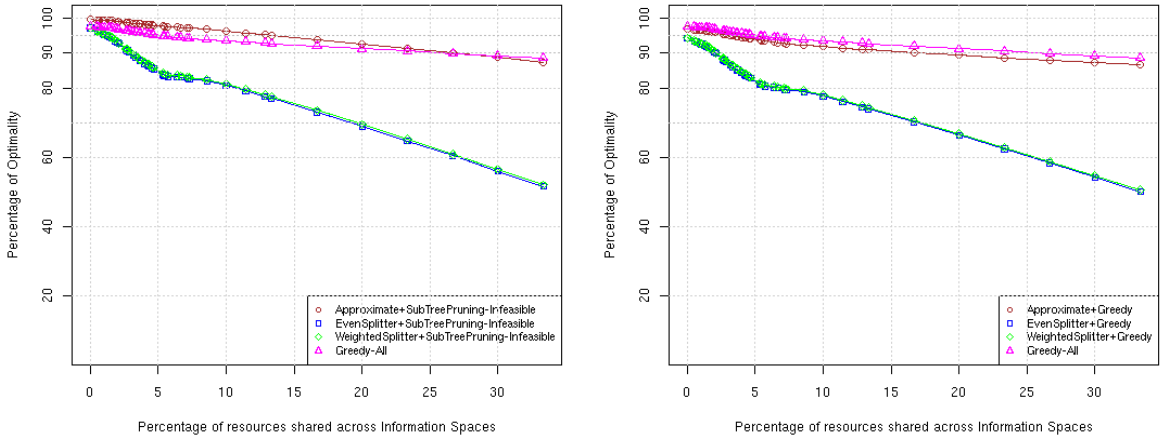


Figure 34: Using "percentage of resources shared across the information spaces" metric for comparing percent of optimality of different inter-information space algorithms run as the first-phase algorithms with (a) Optimizing Brute Force as the second phase, and (b) Greedy Approximation as the second phase algorithms

5.4.5 Comparison of the Contention Metrics

The contention metrics can be grouped based on how they are collected. The first set of these metrics, specifically

1. Percent of infeasible solutions
2. Lowest percent of applications starved in any feasible solution

requires examining the entire space of possible solutions (e.g., by running the Optimizing Brute Force algorithm), a potentially exponential search. They are infeasible to collect in general. However, the rest of the metrics, namely

3. Highest percent of applications requesting the most-shared resource

4. Highest percent of resource requested by applications (in any QoS level) requesting the most-shared resource
5. Percentage of resources shared across the information space

can be collected by a linear or polynomial search of the algorithm inputs.

Put another way, metrics 1 and 2 are measures of the solution space, while metrics 3, 4, and 5 are measures of the scenarios that are inputs to the SRM. This means that metrics 3, 4, and 5 are more susceptible to reflecting characteristics of the scenario generator, especially when comparing these contention metrics against the optimality of the algorithms. Namely, it is possible to craft specific scenarios that would exhibit high contention by metric 3, 4, or 5 and high optimality and other, different specific scenarios that exhibit low contention by these metrics but low optimality, both counter to the results that we saw in the graphs above.

For example, a scenario containing control points with low utility QoS levels that use a large amount of a particular resource and other QoS levels with higher utility but much lower resource usage, should produce highly optimal solutions even though metric 4 would report high contention. Conversely, a second scenario could have very few feasible allocations but with no resource requested more than a few percentage points above what is available. This second scenario would produce a significantly lower value for metric 4, but might produce suboptimal solutions.

Notice that metrics 1 and 2 are not susceptible to this type of manipulation of scenarios, precisely because they are produced from the solution space and not the scenario attributes. In our experiments, we purposely generated a large number of scenarios randomly to cover a wide representative range of characteristics and to avoid generating ones that specifically exhibited any specific characteristics. [25] examines the distribution of the scenarios.

5.4.6 Runtime of Algorithms under Varying Levels of Contention

To measure the runtime of the Dynamic Approximation-Greedy Approximation algorithm, we ran an experiment running the algorithm on a set of scenarios, varying the number of applications in the scenarios, and plotted the runtime against the levels of contention exhibited by the scenarios.

We used the scenario generator to generate sets of 5000 random scenarios each with the number of applications varying from 20 to 100, in steps of 20, for a total of 25,000 scenarios. We set the scenario generator parameters to 3 QoS levels per application, 3 resources used per QoS level, and 110 total resources.

We computed the contention metrics above for each scenario and plotted a regression line for the runtime of each set of scenarios against the level of contention for that set. The number of scenarios and applications makes it unreasonable to gather the first two contention metrics, so we limited this experiment to the contention metrics that can be gathered in linear or polynomial time, i.e., metrics 3, 4, and 5.

Figure 35 illustrates the effect on runtime of changes in metrics 3 and 4, namely,

- Highest percent of applications requesting the most-shared resource

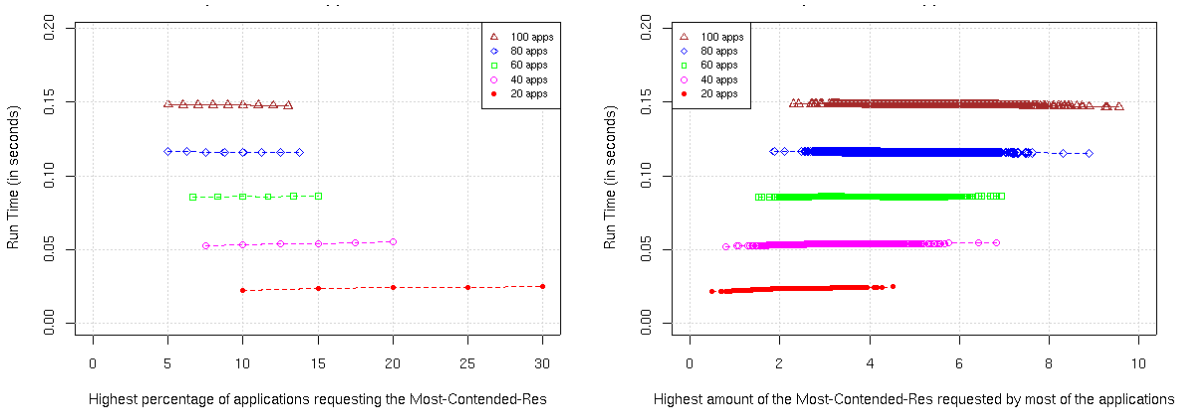


Figure 35: Runtime of Dynamic Approximation-Greedy Approximation as a function of: a) Highest percent of applications requesting the most-shared resource; b) The amount of the most shared resource requested by those applications.

- Highest percent of resource requested by applications (in any QoS level) requesting the most-shared resource

Changes in these metrics do not appear to significantly affect the algorithm's runtime. As the percentage of applications requesting the most shared resource increases, the runtime remains approximately unchanged (Figure 35a). Likewise, as the amount requested of the most requested resource increases, the runtime also remains approximately unchanged (Figure 35b). The declining ranges of the plotted measurements indicate that as the number of applications increases, the percentage requesting any specific resource declines.

Figure 36 illustrates the effect on runtime of changes in metric 5, i.e., the percentage of resources shared across the information spaces. As the value of this metric increases, the runtime increases. However, its impact on the runtime for scenarios with fewer applications is greater than on the runtime for scenarios with more applications. This is because the value of this metric affects the runtime of the first phase of the two-phase algorithm, i.e., the space of applications the first phase runs over gets larger as the value of this metric increases. As the number of total applications increases, the second phase comes to dominate the algorithm's runtime and the impact of an increase in the number of shared resources impacts the total runtime by a lesser amount.

5.4.7 Analyzing the Outliers

While the results of our evaluation of the percentage of optimality of the two-phase approach presented in Section 5.4.2 show good results for many of the scenarios, there were outliers[8] in the

[8] As explained in Section 5.1.3, *outliers* in boxplots represent values that are above 1.5 × IQR + the upper quartile value or less than -1.5 × IQR below the lower quartile value. They are a visualization technique for displaying values that are significantly distant from the cluster of majority of the values. Although outliers can be above or below the IQR, the outliers that we are concerned with are those that lie below the first quartile, as they represent significantly suboptimal solutions.

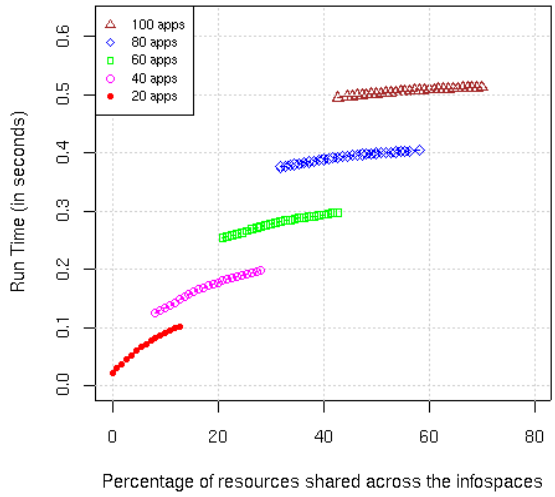


Figure 36: Runtime of Dynamic Approximation-Greedy Approximation as a function of percentage of resources shared across the information spaces

graphs. Specifically, Figure 37 shows that 9.4% of scenarios produced solutions that are considered outliers when Optimizing Brute Force was used for the second phase, and 6.7% of scenarios produced outliers when Greedy Approximation was used for the second phase, from a total of 50,000 scenarios. The number of outliers motivated us to examine the cause that makes the algorithms produce relatively sub-optimal (i.e., outlier) solutions for some scenarios.

Because the results presented in Section 5.4.4 indicated a decline in percentage optimality as levels of contention increase, we started by looking at characteristics that contribute to higher contention, testing the hypothesis that high levels of contention are a cause of sub-optimal allocations.

To examine the contribution of levels of contention on the production of outliers, we examined a subset of the experiments that we conducted which exhibited varying levels of contention. Specifically, we examined the scenarios with 10 applications, 3 QoS levels, and 3 resources per QoS level with total resources varying among 30, 70, and 110 resources. We

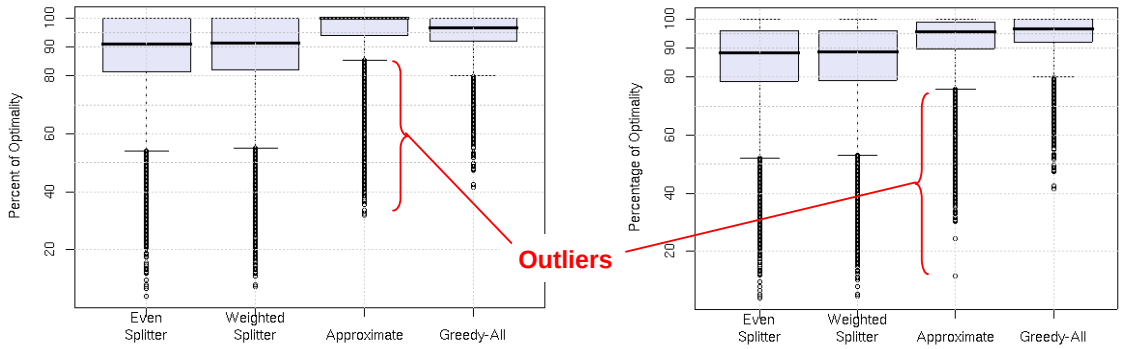


Figure 37: Examining the outliers of the two-phase algorithms with (a) Optimizing Brute Force as the second phase and (b) Greedy Approximation as the second phase.

generated 10,000 scenarios for each level of total resources, which provides us a good basis for representing different levels of contention. The scenarios with a total of 30 resources were highly contentious as the applications had to choose three out of 30 resources. In this case the contention was high because the chance of multiple applications requesting the same three resources from a total of 30 was high relative to those that had a total of 70 or 110 from which to choose three. As we increased the total number of resources from 30 to 70 to 110, the level of contention decreased.

From these sets of experiments, we calculated the median percent of optimality, length of (number of scenarios in) the IQR[9], and the percent of outliers and extreme outliers[10].

The results for the Dynamic Approximation-Greedy Approximation algorithm are summarized in Table 3. As expected, we observed that as the level of contention decreased (from 30 total resources to 110 resources), the median percent of optimality increased. The percent of outliers and extreme outliers also increased. However, the length of the IQR decreased, meaning that the 50% of solutions from the first to the third quartile were more tightly bunched around the median. This means that the increase in the outliers is likely to be artificial, indicating the general increase in percentage of optimality and smaller bunching of results around the improved optimality, but the improved percentage optimality not extending to all the scenarios. This observation is supported by the fact that the percentage optimality of the first outlier also increases as contention decreases, from 61.15%, to 75.75%, to 80.02%, and the percentage optimality of the first extreme outlier increases from 40.71%, to 62.02%, to 68.40%.

Furthermore, the level of contention is not an accurate predictor of the minimum percentage of optimality. While the worst solution in the most highly contentious set (30 resources) is only 11.5% of optimal, which is worse than the worst solution in either of the less contentious sets, the least contentious set (110 resources) actually produces a worse solution than the worst observed solution of the medium contentious set (70 resources). We observed similar results for the Optimizing Brute Force second phase.

Table 3: Trend in median percent of optimality, IQR, and outliers as the number of total resources increases from 30 to 70 to 110 in the two-phase Dynamic Approximation-Greedy Approximation algorithm.

Approx+Greedy	30 Resources	70 Resources	110 Resources
Min % of optimality	11.49425	30.16701	24.48759
Median % of optimality	88.95260	94.66913	96.15848
IQR	13.62601	9.149613	7.744562
1st Outlier	61.14584	75.74697	80.02162
# of Outliers	168	461	642
% of Outliers	0.336	0.922	1.284
1st Extreme Outlier	40.70684	62.02256	68.40478
# of extreme outliers	4	251	495
% of Extreme Outliers	0.008	0.502	0.99

[9] The Interquartile Range in which the middle 50% of values (from the first to the third quartile) lie, as explained in Section 5.1.3.

[10] Whereas outliers lie 1.5 times the size of the IQR from the upper or lower quartile value, *extreme outliers* lie 3 times the IQR from those values.

In order to test whether we could use our contention metrics to predict how the algorithms would perform, we divided the scenarios into two groups: the scenarios that result in greater than or equal to 85% optimality and the scenarios that result in less than 85% optimality. Then we examined the value of the metrics for each of these sets, as shown in Table 4. Most of the values for these two sets are too close to consider as a predictive measure, except for Metric 1 (median % of feasible solutions), which has a wide difference for the two sets (42% versus 12% and 40% versus 15%). However, as shown in Figure 38, the ranges of values for this metric overlaps significantly between the two sets, meaning that knowing the value of this metric for any given scenario does not provide an indication of in which set it is. Therefore it appears as if the contention metrics are not useful for predicting how well either algorithm will perform on a given scenario. That leaves only the total number of applications in a scenario, which affects the runtime, to decide whether to use Optimizing Brute Force or Greedy Approximation as the second phase.

Table 4: Comparison of metric values for scenarios in two sets based on percentage of optimality.

Metric	Approx+Greedy (50,000 total scenarios)		Approx+BF (50,000 total scenarios)	
	≥ 85% (45,538 scenarios)	**< 85% (4462 scenarios)**	**≥ 85% (45,477 scenarios)**	**< 85% (4523 scenarios)**
1. Median % of feasible solutions	***42%***	***12%***	***40%***	***15%***
2. Highest % of non-starved applications	100%	90%	100%	90%
3. Max % of apps requesting the most shared resource	20%	30%	20%	30%
4. Max amount requested of most shared resource	1.5x	1.8x	2.4x	6.0x
5. % of resources shared across the information spaces	2%	7%	2%	6%

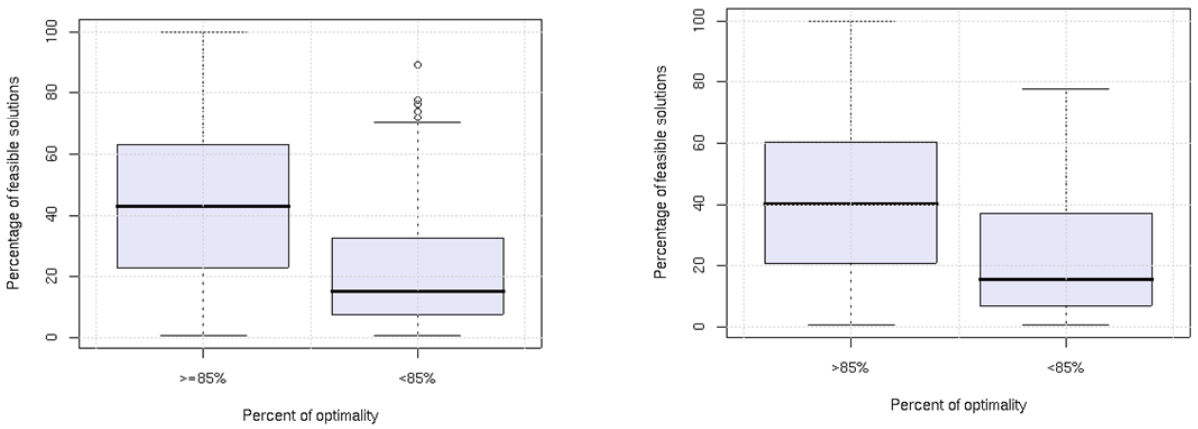


Figure 38: Overlap in two categories (≥ 85% and < 85%) of scenarios for the values for percent of feasible solutions for (a) Dynamic Approximation-Greedy Approximation and (b) Dynamic Approximation-Optimizing Brute Force.

6 The Prototype System Resource Manager Version 2

We developed an enhanced prototype of the QMS System Resource Manager that utilizes the new QoS management algorithms. The enhanced SRM allocates QoS levels and resources associated with these QoS levels to applications that share resources either *within* an information space or *across* multiple information spaces. The enhanced SRM prototype employs a *two-phased* algorithm for multiple information spaces. Based on the results of the experimental evaluations described in Section 5, we prototyped the *Dynamic Approximation* algorithm as the inter-information space first phase, and the *Optimizing Brute Force* and *Greedy Approximation* algorithms as options for the intra-information space second phases. The resulting SRM prototype is able to allocate QoS levels to large numbers of applications across multiple information spaces, in a dynamic environment where new applications join, existing applications leave, the roles and priorities of applications change, and the priorities of information spaces change.

In the following sections, we describe the enhanced System Resource Manager and other prototype components that we developed to work with it.

6.1 The Enhanced SRM Prototype

The enhanced SRM uses the two-phased approach to QoS allocation for multiple information spaces described in Section 4.5 and illustrated in Figure 39. The first phase runs an inter-information space algorithm to identify the resources shared between information spaces and divide them between the information spaces. The resources allocated by the first phase become the total resources available to each information space for the second phase. An intra-information space algorithm is then run in each information space (constrained by the results of the first phase) to allocate QoS to the applications in that information space.

The following describes the primary features of the enhanced SRM:

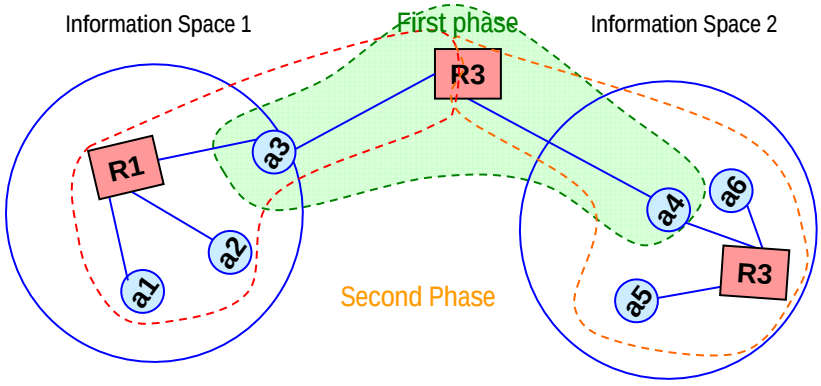


Figure 39: A two-phase strategy that the SRM uses for providing allocations. In the first phase, the SRM allocates resources to applications that share resources across information spaces. Using these allocations as constraints, in the second phase the SRM allocates QoS levels and associated resources to applications within each information space.

- *Pluggable algorithms* – To create the enhanced SRM, we reused the core code from the simulator that invokes the QoS allocation algorithms in the experiments described in Section 5. This results in a *pluggable architecture* for the new SRM. New algorithm implementations can be plugged into the SRM without changing the code that passes the arguments and invokes the algorithms, enabling the SRM to be extended as new algorithms are developed.
- *Thread-safety* – We wrap the SRM QoS allocation algorithm invocation in a *mutex* operation so it is thread safe even if the SRM receives messages initiating QoS allocation requests while it is still processing the previous allocation request.
- *Synchronize distributed QMS components* – One of the challenges facing the layered QMS architecture is ensuring that the elements of QMS are operating under the guidance of a consistent policy. To help achieve this, we added a *start-time* to all of the control messages[11] going to or from the SRM. Instead of acting immediately upon a control message (a situation that could lead to race conditions), each QMS component will wait to act upon a control message until the start time. Combining this with synchronized clocks using NTP[12] and the handling of intermittent communications described next, this design increases the likelihood that SRMs and LRMs will be acting upon consistent policy.
- *Handle communication disruptions* – Since the intermittent communications in tactical environments or other network problems can result in disconnections in the network, varying link-capacities, or link losses, we cannot guarantee messages from an SRM component will reach its recipients (LRMs, QoS internals display, etc.) reliably or within a specific time. Therefore, we designed and implemented the new SRM prototype to gracefully handle control messages not reaching their destinations and ensure consistent policy enforcement and view of context (i.e., inputs to the SRMs) throughout a set of information spaces. To ensure this, we designed and implemented the SRM to do the following:

    a. *Provide each message with a message identity number (msgID), and time-to-live parameter* – To ensure that an LRM or SRM doesn't process a message that was delayed during transmission or delivered out of order, we provide each control message sent or received by the SRM with a message ID and a time-to-live parameter. If an LRM or SRM receives a message whose msgID is older than the msgID of the message that it is currently handling, the message is disregarded. Likewise, if the time-to-live parameter on the message has expired, the message is disregarded.
    b. *Periodically send allocations and QoS policy messages* – Since the control message now include an expiration time (the time-to-live parameter), valid

[11] *Control messages* carry control information such as QoS policies or resource allocations. Contrast these with the data messages that carry MIOs and payload (in an information space context).

[12] NTP is a Network Time Protocol used to synchronize distributed clocks (http://www.ntp.org/, http://en.wikipedia.org/wiki/Network_Time_Protocol).

control messages (such as the current QoS policy), need to be periodically retransmitted. The SRM includes a configurable timer that tells it how frequently to send the control messages. At every timeout period for this timer, the SRM sends the allocation and QoS policy messages. The start-time, time-to-live, and retransmission times need to be chosen to be consistent with one another, so that the start-time provides enough time for every connected LRM to receive a new QoS policy before any LRMs act upon the policy, so that the current (or new) policy is received before the old one expires, and so that any LRM that does not receive a new policy is treated as disconnected and stops acting upon an old policy (i.e., its policy expires).

c. *Hold the last message* – The SRM (and LRM, although we have not yet implemented this) holds the last control message it received and processes a new message only if its contents are different from the last one. This eliminates the overhead of processing control messages that are periodically resent (as opposed to new control messages) and reduces the chance of thrashing in the system. Each valid control message, duplicate or not, will reset the time-to-live clock.

The SRM is designed to be run in a distributed manner. The three inter-information space algorithms that we describe in Section 4.5.1 as first phase choices (including the one that we prototyped in the SRM, Dynamic Approximation) are deterministic. Therefore, the SRM for each information space can run both the first and second phases of the two-phase algorithm in parallel. Given the same inputs, the SRM for each information space will reach the same result from the first phase, and use it as input to the second phase.

For the SRM demonstrated and delivered to AFRL on November 15, 2007, we implemented the following combinations of first- and second-phase algorithms:

- Dynamic Approximation + Greedy Approximation
- Dynamic Approximation + Optimizing Brute Force

6.2 SRM Interfaces

This section describes the data structures and interfaces for the enhanced SRM.

The following structure provides metadata for control messages sent to QMS components, such as messages from the Mission Manager to the SRM or messages from the SRM to LRMs:

```
module qms {
  struct MsgMetaData {
   //! ID for each msg to differentiate it from previous msgs
   long msgID;
   //! Time at which the msg was generated
   double generationTime;
   //! how long the msg can live with respect to its generation
time
   double TTL;
```

```
    //! Time at which we want the QMS component to start
executing
    double startTime;
   };
  };
```

The metadata structure includes the information needed to synchronize control messages and handle disruptions in communications, as described above, including a message ID, the time the message was generated, the time-to-live for the message, and the time at which the recipient should act on the message.

The following structure identifies the algorithms that the SRM should use for its first and second phases:

```
struct Algorithms
  {
    //! first phase algorithm
    string firstPass;
    //! second phase algorithm
    string secondPass;
  };
```

The following structure and sequence allows a set of applications to be described for the SRM. This only describes the applications. It does not include enough information to allocate QoS levels or resources.

```
struct ApplicationStr
  {
    //! name of the application - app-1, app-2
    string appName;
    //! type of application - TACP, UAV.
    string appType;
    //! Information space in which this application is
operating
    string coi;
    //! role that this application is playing - may vary
depending upon the appType
    string role;
  };
typedef sequence<ApplicationStr> ApplicationStrSeq;
```

The following interfaces are used to *invoke*, *initialize,* and *configure* the SRM. Invoking the SRM initiates a QoS allocation. Initialization is a special case of the configuration process in which an SRM is started up with an initial set of applications comprising some number of applications for each information space. The configure interfaces enable choosing the algorithms

that the SRM will run, adding new applications in bulk, removing applications, or changing the role of existing applications. The following paragraphs describe these interfaces.

Invoke the SRM to initiate resource allocations. The following interface initiates QoS and resource allocations by invoking the functions for the first-phase and the second-phase algorithms.

```
oneway void invoke(in MsgMetaData header);
```

Initialize the SRM with a set of applications. The following interface sends an application set as an initial state to the SRM.

```
oneway void initialize(in MsgMetaData header, in
ApplicationStrSeq apps);
```

Bulk addition of applications. The following interface enables a set of applications to be added in one command (useful for experimenting with the SRM at scale).

```
oneway void addApps (in MsgMetaData header, in
ApplicationStrSeq apps);
```

Change the role of an existing application. The following interface enables the role of an application ("appName") to be changed to a new role ("newRole").

```
oneway void changeRole(in MsgMetaData header, in string
appName, in string newRole);
```

Remove an application. The following interface enables the application whose name is "appName" to be removed. As a consequence, the SRM is no longer responsible for providing allocations for the specified application.

```
oneway void removeApp(in MsgMetaData header, in string
appName);
```

Choose first and second phase algorithms. The following interface enables selection of the algorithms that the SRM will run for the first and second phases.

```
oneway void chooseAlgorithms(in Algorithms alg);
```

6.3 Prototype Mission Manager

For the enhanced SRM prototype, we updated the prototype Mission Manager software to serve as a driver for operating the enhanced SRM. The prototype Mission Manager provides a graphical user interface and sends commands to the SRM using the interfaces described above. As shown in Figure 40, it supports initializing the SRM, adding new applications, removing

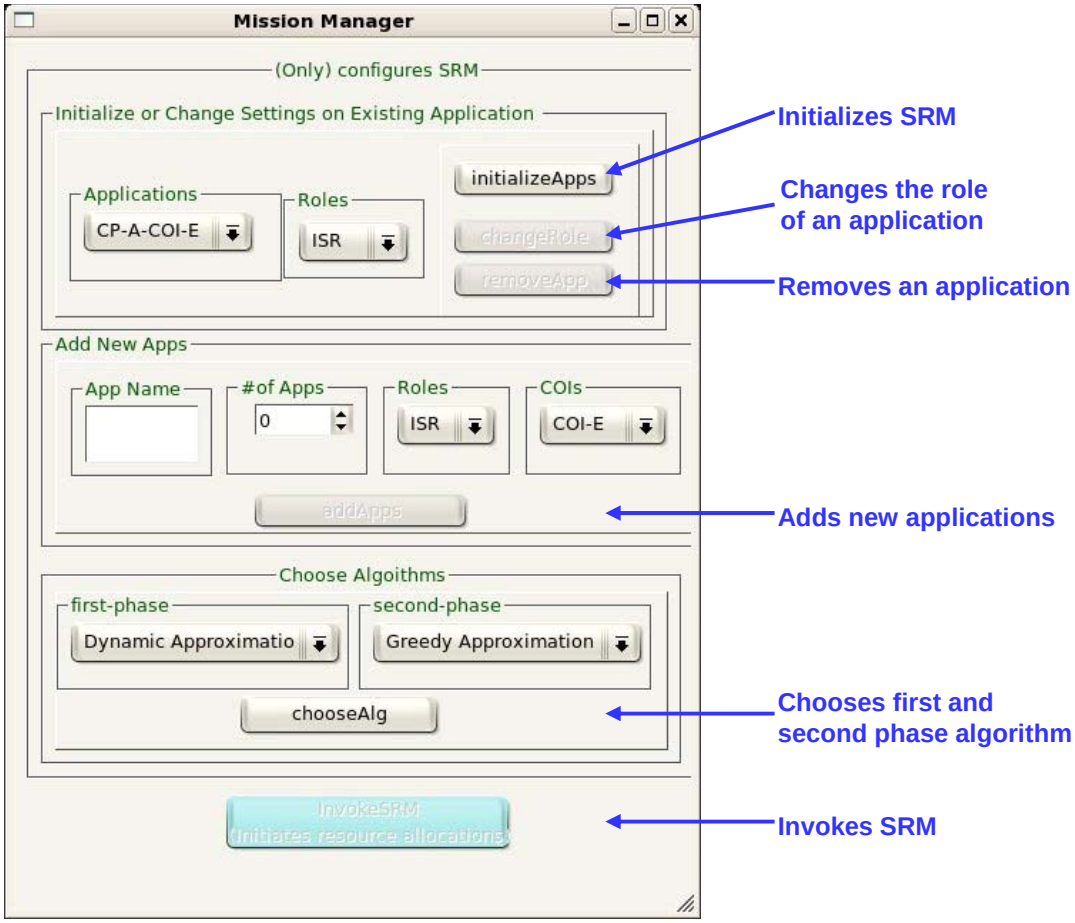


Figure 40: The version 2 Mission Manager prototype, used for exercising the SRM

existing applications, changing the role of an application, choosing the algorithms that the SRM runs, and invoking the SRM.

6.4 Prototype Support Software for the Enhanced SRM

The SRM prototype needs configuration information about the number of information spaces, applications, their QoS levels, utilities, and resource usage. Ultimately, some of these will be provided by runtime calculation and monitoring, configuration files or scripts, or third party capabilities. In the current prototype, we provide the configuration and input information that the SRM needs through a combination of constructor settings, command line arguments, configuration files, and support software. This section describes each of these, including the following:

- Configuration and initial setup

- Resource Mapper
- Internals' Display

6.4.1 Configuration and Initial Setup

The current SRM prototype has the set of information spaces (i.e., the COIs), the set of resources each uses, and the set of resources that they share specified in the SRM constructor function. Ultimately, this information should be provided by an information space administration tool, resource discovery service, or configuration file.

6.4.2 Resource Mapper

The resource mapper is a component envisioned to determine the resources used by applications in particular roles and at particular QoS levels at runtime. In the current prototype software we provide a version of the ResourceMapper interface useful for demonstrating the SRM functionality. This version, called RandomResourceMapper maps an application's resource usage in a particular QoS level and role to a random selection of resources from the resources available. It is invoked in the SRM constructor. The RandomResourceMapper takes as input a set of "Typename, Filename" pairs, which identify configuration files for each type of application. Each application type configuration file provides information about the applications in each COI, their QoS levels, utilities, and resource usage. Figure 41 shows an example application type configuration file.

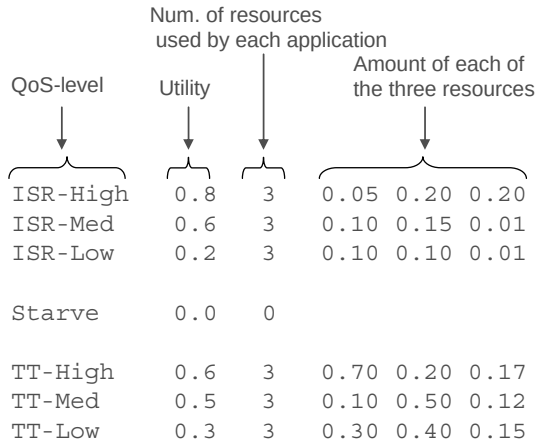


Figure 41: The configuration file read by the Resource Mapper.

6.4.3 Internals' Display

The Internals' Display is a graphical user display that shows information about the operation of the SRM algorithms. This display depicts the number of applications in different information spaces, the resources shared by the applications across the information spaces, the runtime of algorithms in different information spaces, the allocation produced by the SRM, and (when possible) the optimality of the resulting allocation.

This display has the following six panels as illustrated in Figure 42:

A. A progress bar that reflects the progress of the SRM's execution. In the current prototype, the progress bar moves at the completion of each stage of the SRM execution.
B. The total number of applications in all the information spaces and the number of applications in each individual information space.
C. The execution time of the first- and second-phase algorithms.
D. The number of resources shared across the information spaces and the number of applications sharing those resources. It also displays the total number of resources available and the total number of applications.
E. A comparison of the sum of the utilities of the allocations provided for both information spaces with the utility of the optimal allocation. The optimal utility is computed by running the Optimizing Brute Force algorithm on the combination of both information spaces. This panel reports "scenario too big", if the search space is too large (greater than 3^{28} possible allocations).
F. A representation of the solution returned by the SRM. The first two rows display the aggregate allocation (across both information spaces) compared to the optimal allocation (if available). The second two rows display the allocations for each information space. The first three columns of each allocation show the number of applications that received the "High-QoS", "Medium-QoS", and "Low-QoS" choice appropriate for their role (these QoS levels come from the configuration file in Figure 41). The next column ("NRA") indicates the number of applications that received a "Non-Role Appropriate" allocation, i.e., the application received an allocation of resources but not associated with any of the QoS levels requested for the application's role. The last column represents the number of applications that got starved, i.e., they received no allocation.

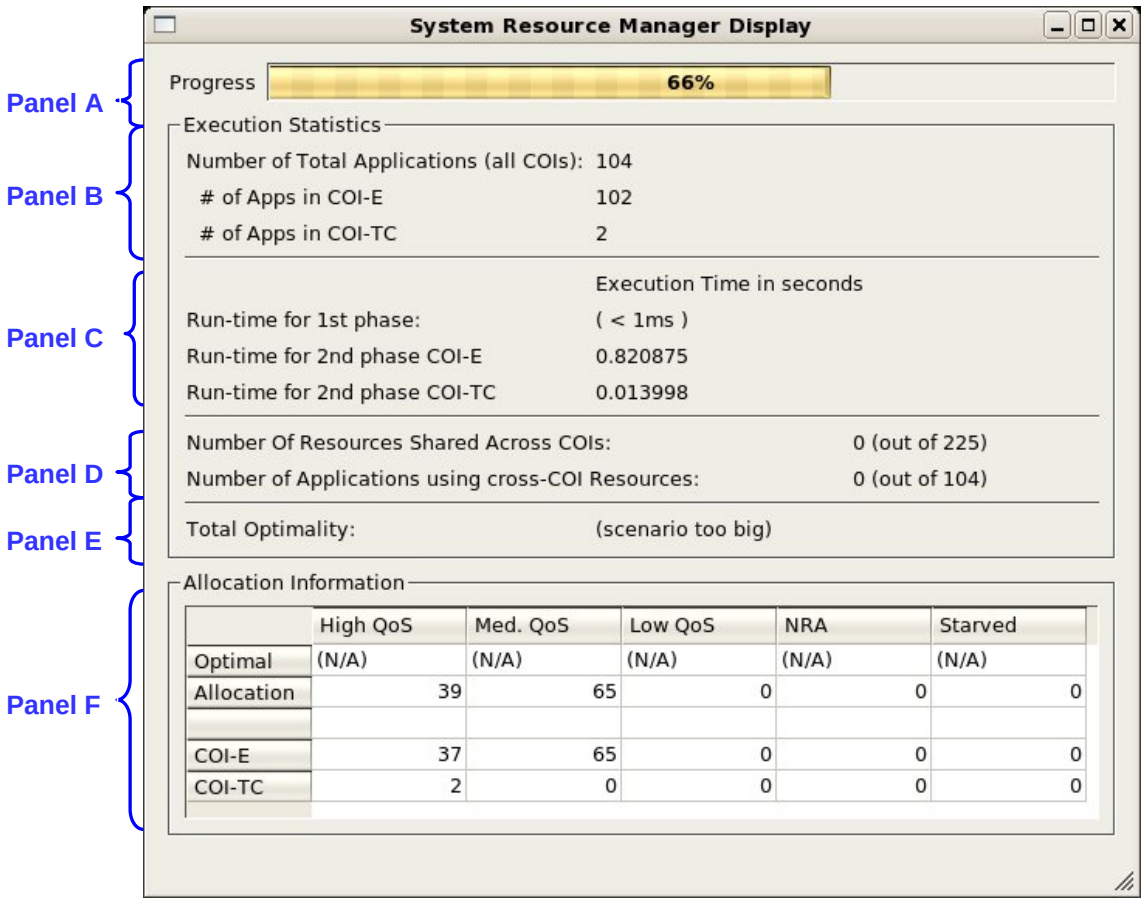


Figure 42: The Internals' display for the SRM.

7 Demonstrations

We conducted several demonstrations of our software prototypes throughout the DynRIIC project. In this section, we describe three of the demonstrations in more detail.

7.1 Demonstration of QMS Version 1 (TIM at AFRL, October 2, 2006)

In this section, we describe a demonstration we conducted of a simulated time sensitive targeting (TST) operation, how QMS provides its QoS management, and a comparison of performance of the system with and without QMS active. We conducted this demonstration at a technical interchange meeting (TIM) at AFRL on October 2, 2006.

7.1.1 The Time Sensitive Targeting Demonstration Scenario

The TST demonstration scenario consists of two communities of interest, ISR COI, an institutional COI, and TST COI, an expedient COI. The information space for the ISR COI uses the JBI RI for application data. The information space for the TST COI uses OCI's DDS implementation for its application data. Both information spaces use the CORBA Notification Service for control traffic.

There are three simulated UAVs and three simulated C2 processes. The simulated UAVs are information publishers, sending imagery information. The simulated C2 processes are information subscribers, each receiving and displaying imagery from one of the simulated UAVs. Each UAV-C2 receiver client pair can operate in one of three roles:

- *Surveillance (ISR)* where the UAVs simulate performing routine surveillance of an area of interest. The QoS requirements are that the imagery should be at sufficient rate that there are no gaps in coverage and that it should be at least of sufficient resolution to detect whether there is a potential target that merits a closer look. This role is the least important relative to the other roles.
- *Target tracking (TT)* where the UAVs have been instructed to take a closer look at (i.e., track) a potential target. This is a role in which a commander has decided that there is a potential threat or target in surveillance imagery, and the item of interest should be carefully observed. The imagery should be high resolution and at a sufficient rate to track a moving object. This role is the most important of the three roles.
- *Battle damage assessment (BDA)* where an action has taken place, such as engagement of a target, and a UAV has been directed to collect imagery for assessing the effectiveness of the action, such as the extent of damage. The imagery should be high resolution and a large size to facilitate off-line analysis, but does not have to be at a high rate. This role is more important than routine surveillance, but not as important as target tracking.

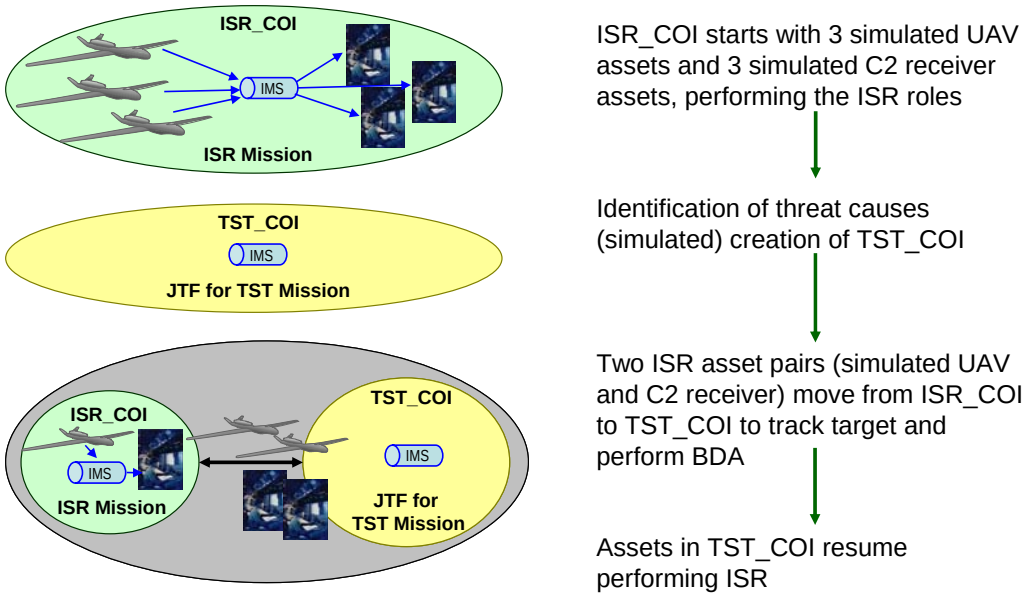


Figure 43: The TST demonstration scenario

As illustrated in Figure 43, the demonstration scenario consists of the following sequence of events:

1. Participants in ISR_COI are gathering and analyzing intelligence imagery (ISR role). A commander at the C2 detects suspicious insurgent activity in an area of interest (AOI).
2. The mission manager creates a new TST_COI to act on the suspected insurgent activity. We simulate the creation of a new information space, which results in dynamic deployment of two new SRMs – one for TST_COI and one for the resources shared between TST_COI and the existing ISR_COI.
3. The mission manager requests a UAV from ISR_COI to relocate to TST_COI and tasks it to perform surveillance in the AOI. Based on the imagery, he reassigns the UAV to track a target and engage it (TT role). This results in resource reallocations and QoS adaptations in both information spaces.
4. On completion of the engagement, the mission manager requests another UAV from ISR_COI to move to TST_COI and collect battle damage imagery (BDA role). This results in another reconfiguration by the QMS.
5. On completion of both tasks, the two UAVs in TST_COI are tasked to move to the ISR role, with another resulting reconfiguration by the QMS.
6. As the final step in the demonstration, the QMS system is turned off to show the performance of the demonstration components and COIs when behavior is unconstrained.

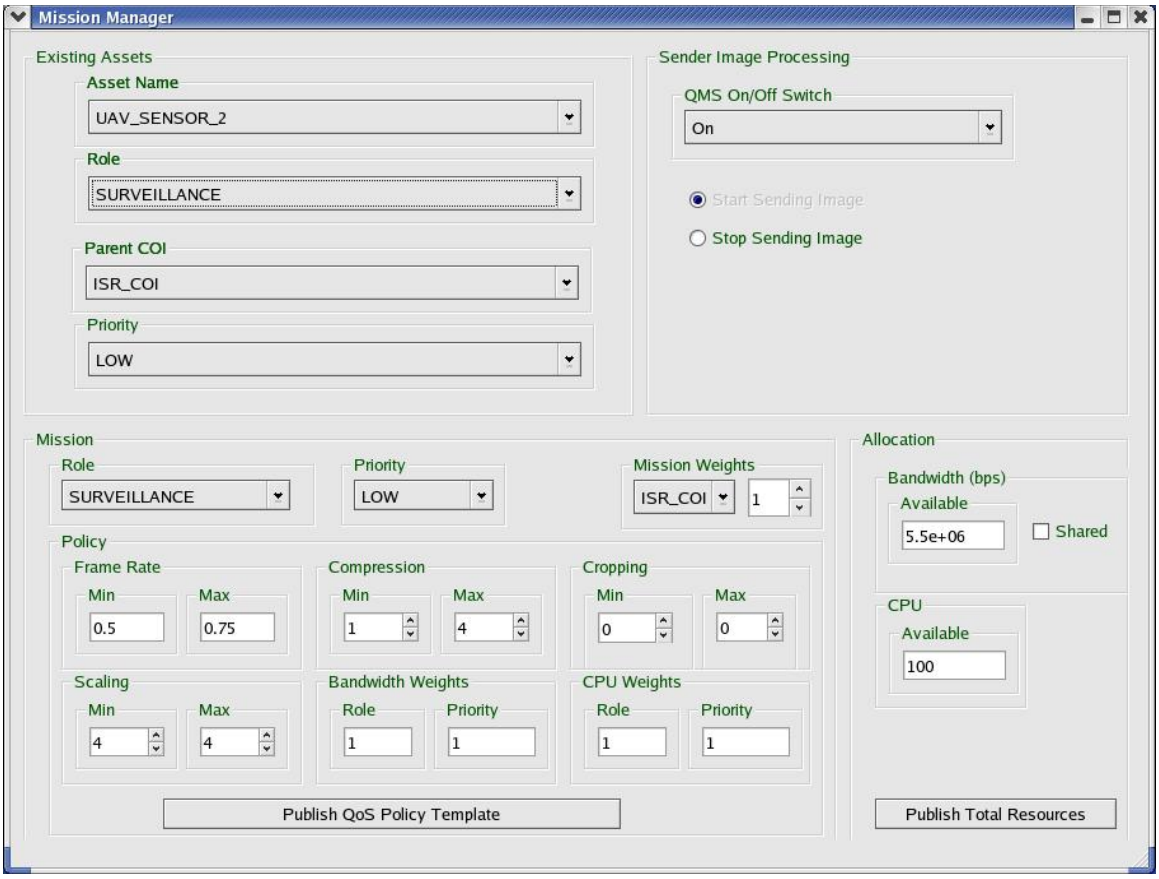


Figure 44: The Mission Manager used to control the steps in the demonstration.

7.1.2 Execution of the Demonstration

A detailed description of the demonstration and instructions for running it are included in [1]. The Mission Manager user interface, illustrated in Figure 44 and described in Section 3.4.1, was used during the demonstration to move assets (and their client software) between COIs, to change the role of participants, to turn QMS on and off, and other demonstration driving operations.

The QoS internals display, illustrated in Figure 45 and described in Section 3.4.2, showed the internal behavior of the system. It displayed the COI to which each asset belonged, the IMS or pub/sub middleware servicing each COI, the role of the asset, its priority, the amount of resources allocated and used by the participant, the QoS policy, and the actual delivered QoS.

The changes in roles and delivered QoS were visible to demonstration observers by observing an application-specific display of the imagery being published and consumed by the demonstration participants. A sample of the imagery displays is illustrated in Figure 46. The color of the border around each display indicated the role of the producer and consumer of the imagery (blue for the surveillance role, green for target tracking, and yellow for battle damage

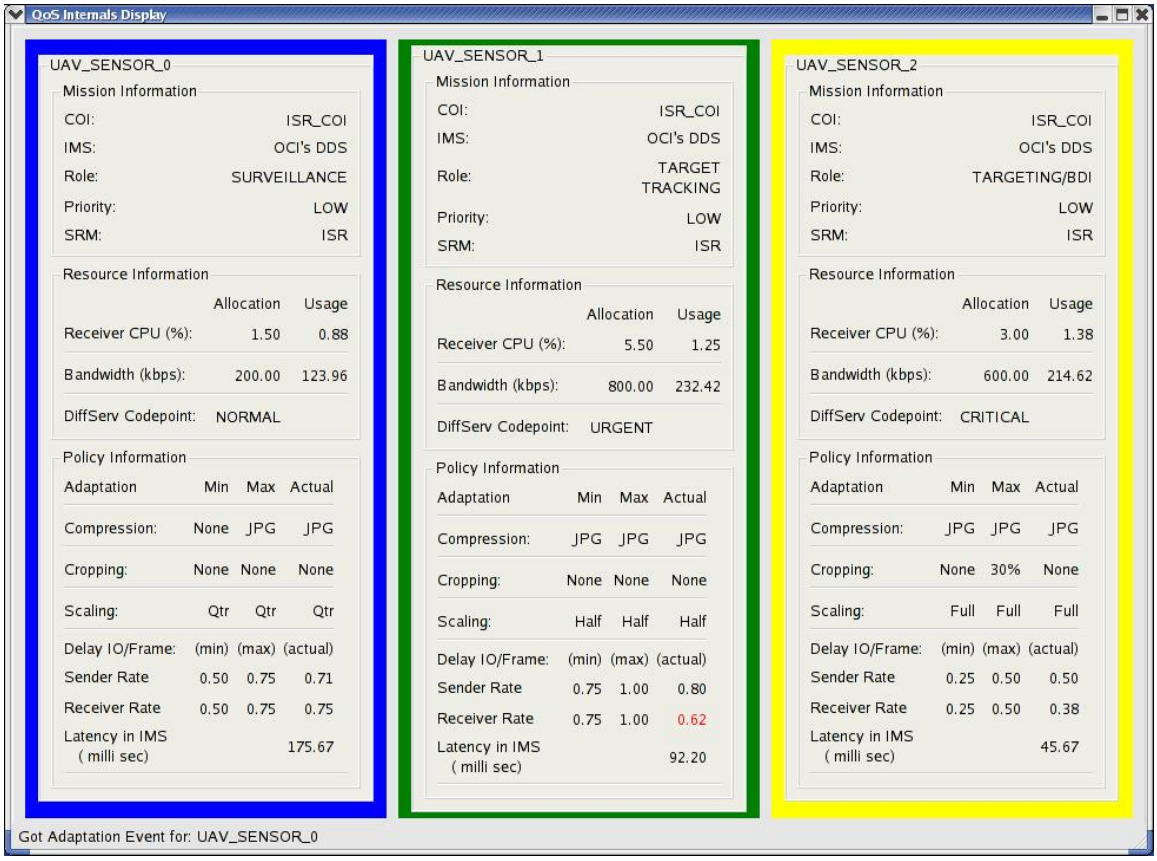


Figure 45: The display of the QoS policies and QoS-related behavior of demonstration participants.

Figure 46: The display of imagery data upon delivery by the demonstration system. The colored borders indicate roles of the publisher and subscriber. The differences in imagery quality, size, and rate were as a result of QoS management.

indication). While some of the QoS adjustments (e.g., compression) might not have been noticeable to the naked eye, others (e.g., rate change, scaling, and cropping) were readily apparent from the display.

Figure 47 shows the performance of one of the demonstration participants, UAV_Sensor_1, a client publishing imagery. The client starts off in the ISR role in the ISR_COI, which uses the

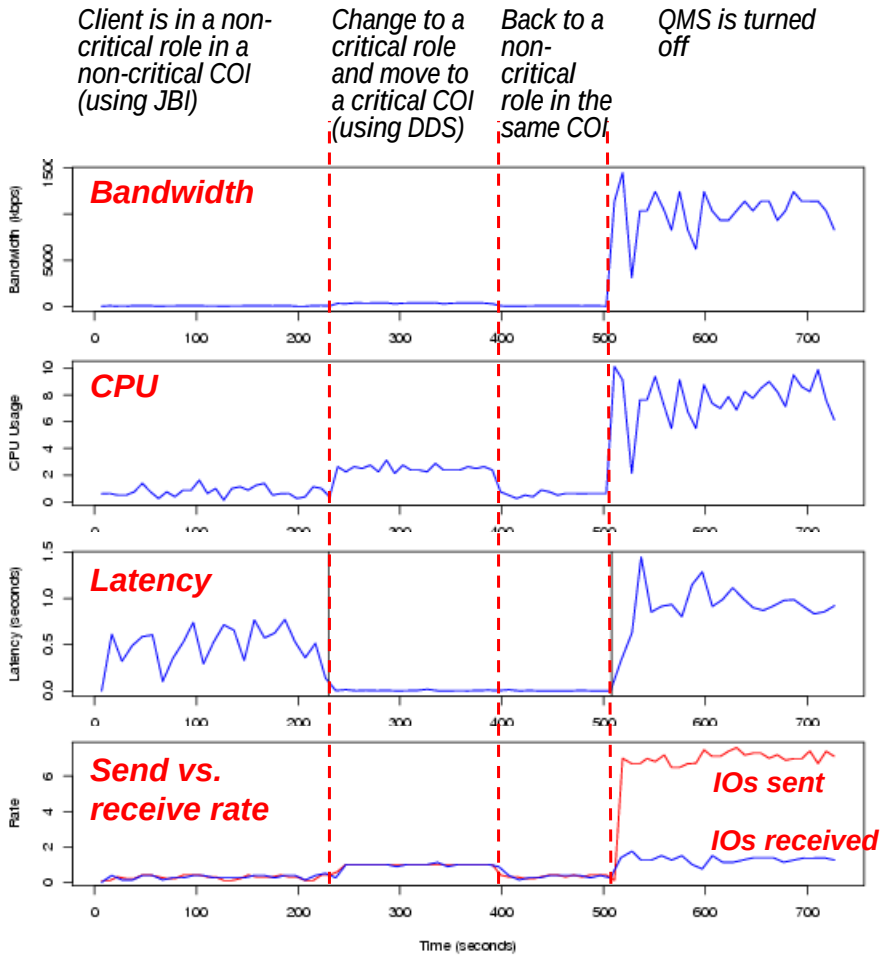


Figure 47: Performance of UAV1 in the demonstration.

AFRL JBI RI 1.2.6. The QMS system divides bandwidth and CPU among all of the clients based on the importance of their role in the mission, and shapes the clients' information production and processing to meet its mission requirements within the allocations. In this case, the UAV_Sensor_1 client is the same priority as other surveillance clients and receives enough bandwidth and CPU to deliver imagery at medium resolution and at a rate to get sufficient coverage. This results in the QMS system shaping the information by compressing the imagery and controlling the rate. It also sets the proper parameters on the underlying platforms so they can properly prioritize the traffic. In this case, the AFRL RI does not have differentiated service, so there is some introduced latency and jitter, but no data loss and controlled resource usage.

The first red line indicates moving UAV_Sensor_1 to the TST_COI and setting its role to *target tracking*. The target tracking role has a higher priority than the other roles, and the TST_COI has a higher priority than the ISR_COI. This causes this client to have a relatively higher priority than before and a need for higher QoS to transmit a higher rate of imagery. The QMS system allocates more bandwidth and uses it to send imagery at a higher rate, 2-3 times

higher than before. The TST COI uses the OpenDDS, which provides differentiated service and responds to the QMS control to maintain no data loss and low latency.

The second red line indicates changing the role of UAV_Sensor_1 back to ISR. The client is back to a non-critical, ordinary priority, and its need for high rate imagery is reduced. The CPU usage and send rate decline.

The third red line indicates that we turned off the QMS system to show how the unconstrained, uncontrolled system would operate. All the clients are sending unconstrained, regardless of their relative priorities and their mission needs, and competing for the available bandwidth and CPU. While the UAV_Sensor_1 client is getting more bandwidth and CPU by chance, its use is unconstrained and jittery. It and other clients are not getting a reliable amount of resources, and 2/3 to 3/4 of the IOs being sent are lost and those that are getting through are experiencing significant and unpredictable delay.

7.1.3 Performance of the Demonstration Software

This section shows more details about the relative performance of the participants in the demonstration. The results described in this section were gathered by running the demonstration twice, once with QMS active and once with QMS inactive, and collecting metrics on latency, data loss, and resource usage. We ran the demonstration on three 2.4 GHz Intel Pentium laptops with 512 KBRAM connected by a 100Mbps Ethernet switch. One of the machines hosted all three simulated UAVs on RedHat Linux 9.0. A second machine hosted all three simulated receivers and OpenDDS on RedHat Linux 9.0. The third machine hosted the JBI RI IMS on Fedora Core Release 3. We synchronized the clocks on all three machines using NTP. Data flowed from the simulated UAVs to the IMSs and then to the simulated receivers.

Figure 48 shows the end-to-end latencies for information delivery with and without QMS active. With QMS active, the end-to-end latency of imagery delivery is nearly zero with very low variance for all the simulated UAVs: UAV0 performing ISR, UAV1 performing TT, and UAV2 performing BDA. In contrast, without QMS, the AFRL RI being used for information from

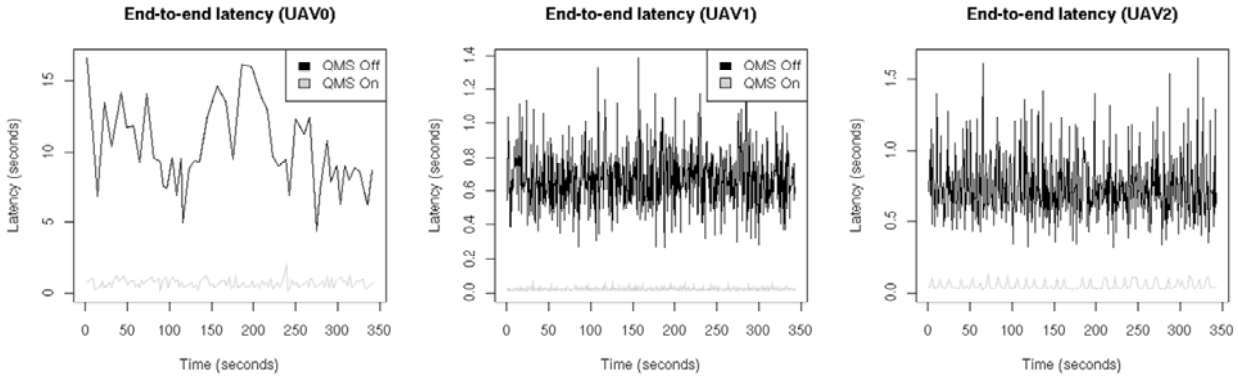


Figure 48: Latency of information delivery in the demonstration with and without QMS active. Without QMS, there is high variance and significant introduced latency, whereas with QMS the imagery is delivered with near zero delay.

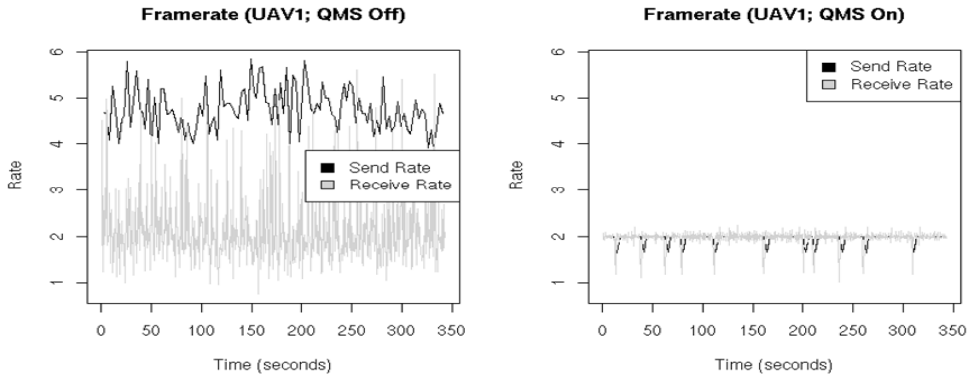


Figure 49: (a) Without QMS, UAV1 performing target tracking and using DDS experiences high variance and information loss. (b) With QMS, the sending of TT imagery tracks the receipt very well, with predictable information delivery.

UAV0 introduces uncontrolled delay, with information objects being delayed by as many as 15 seconds. DDS, being used in the TST_COI for UAV1 and UAV2, performs better with a worse case delay of less than 2 seconds, but still with significantly more jitter.

DDS avoids introducing as much delay into the information delivery by dropping information. As Figure 49(a) illustrates for UAV1 performing the TT role, DDS is simply not able to keep up with the information rates that the information sources can provide, even at a modest five images per second. Without QMS management, the data loss is uncontrolled and highly variable. Figure 49(b) shows that QMS recognizes the DDS as a bottleneck and manages it by throttling the send rate back (and using it for the most important imagery), resulting in controlled, predictable behavior.

Figure 50 and Figure 51 show how well QMS controls resource access and usage, and how QMS allocates resources to clients based on their role in the mission. Figure 50 shows network usage on the link from the IMS and dissemination service to the receivers. Network usage varies widely for all the simulated UAVs with QMS disabled, regardless of the importance of their

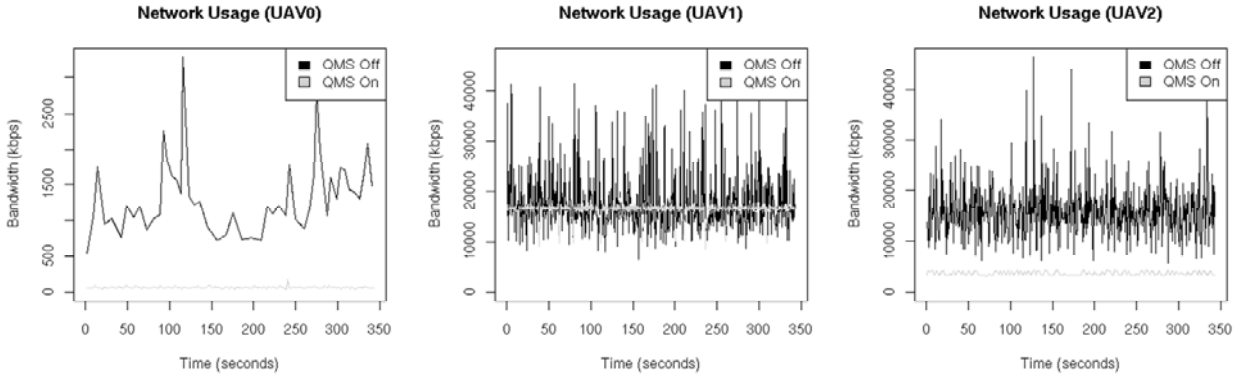


Figure 50: Without QMS, network usage is best effort, resulting in bursty, unpredictable network usage. With QMS, UAV1 performing TT is allocated the most, UAV2 performing BDA next, and UAV0 performing ISR next. The behavior of each is managed to effectively use the allocated bandwidth according to the role, resulting in predictable, effective bandwidth usage.

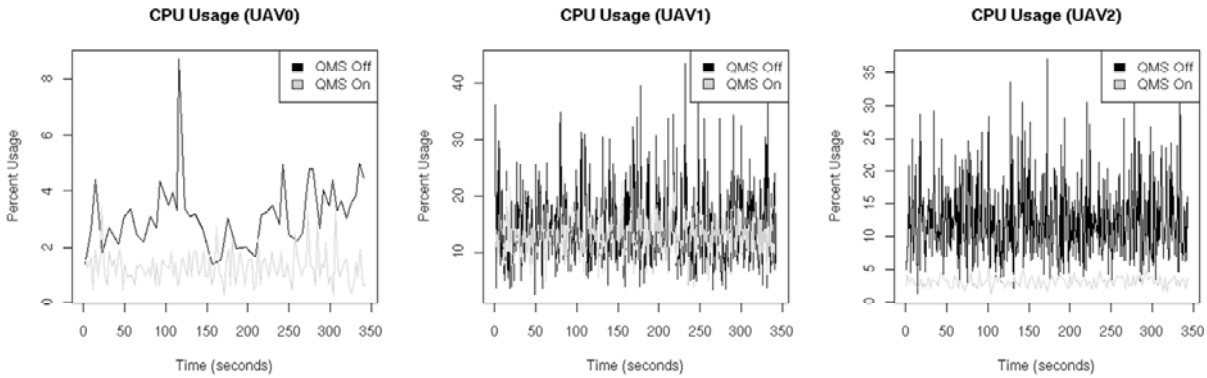


Figure 51: QMS provides less bursty, more effective use of CPU resources based on the roles.

roles. The worst case network usage can reach over 40 Mbps for the information streams in the TST_COI but, as described above, the RI and DDS are unable to sustain the data rates needed for that network usage, resulting in bursty behavior, uncontrolled loss, and unpredictable resource usage. In contrast, with QMS enabled, more network bandwidth is allocated to the TT role (UAV1) than to the BDA role (UAV2), which in turn gets more than the ISR role (UAV0); and usage of the bandwidth is controlled to ensure predictable, sufficient QoS.

Figure 51 shows similar results for CPU usage at the simulated C2 node hosting the simulated receivers. In this case, while the CPU is part of the overall end-to-end QoS management capability being provided by QMS, it is not the bottleneck being managed (the IMS and dissemination service are the main bottlenecks). Even so, Figure 51 illustrates that uncontrolled usage can be bursty, approaching significant fractions of the available resources in the worst case, and does not distinguish between the roles. In contrast, using vanilla Linux priorities, active QMS management reduces the jitter, increases the predictability, and ensures that available CPU is used in accordance with the mission priorities.

7.2 2007 PI Meeting Demonstration (OIM PI Meeting, Washington, DC, April 24, 2007)

We conducted a demonstration of our prototype software at the Operational Information Management Principal Investigators meeting held in Washington, DC, April 24-25, 2007. The demonstration was similar to the previous one, but showcased the allocation of QoS levels using multiple resources, using the approximation algorithm described in Section 4.3.

Figure 52 illustrates the basic operation of the demonstration. It started with two information spaces. The first information space had two publisher clients and one subscriber client and the second information space had one publisher client and one subscriber client. The publisher clients simulate UAV image sensors and the subscriber clients simulate command and control (C2) centers.

The demonstration showcased the allocation of QoS levels and their resources among the control points in each information space. It then migrated one of the publishing clients from the first to the second information space and illustrated the QMS system reallocating the QoS levels

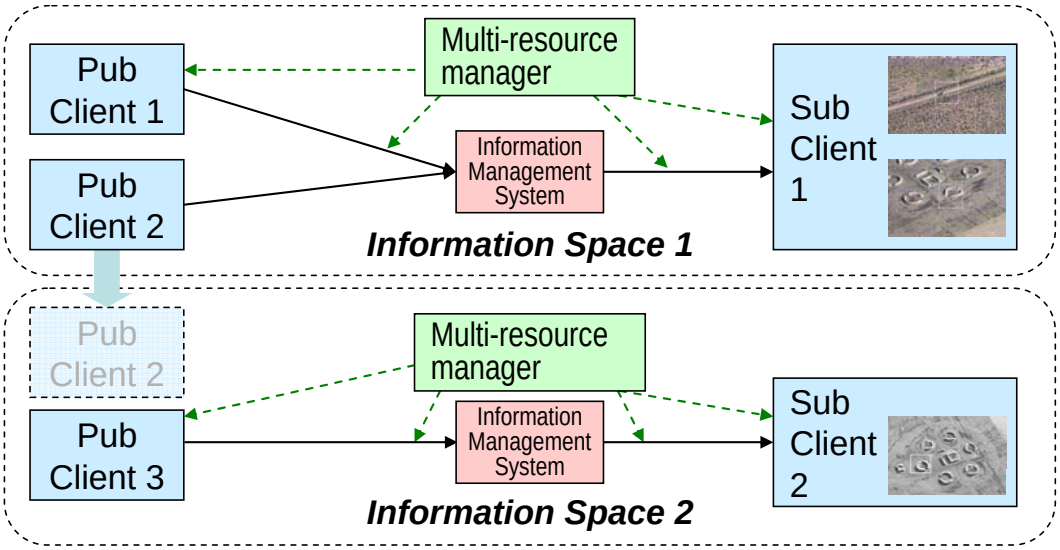


Figure 52: Basic operation of the OIM PI meeting demonstration

and resources dynamically. The demonstration showed the QMS reallocating QoS levels as roles of the clients change (from ISR to TT and from ISR to BDA). Finally, the demonstration illustrated the relative benefit of QMS, by turning off the QMS system and showing the relatively lower quality and less predictable behavior of the system, in the form of increased jitter and uncontrolled information loss.

We conducted the demonstration on four laptops, with the software distributed as illustrated in Figure 53. This display, which was available throughout the demonstration, also shows the topology of the demonstration from the SRM point of view. It shows the QoS levels allocated to each control point (CP-A, CP-B, and CP-C) and the resource allocation to which each allocated QoS level corresponded.

As in the previous demonstration, the simulated operational behavior was readily available to observers through a simulated C2 display, illustrated in Figure 54. This display showed the following:

- Imagery for each subscriber client
- The client publishing the imagery (i.e., UAV1, UAV2, or UAV3)
- The role of the publishing client, presented as both a colored boundary for the image and by name
- The rate that imagery is being sent (i.e., published)
- The rate that imagery is being received by the subscriber
- The latency for the imagery

Observers could see changes in the allocated QoS levels by observing changes in the rate, size, and resolution of the imagery, as well as the display of the QoS level in Figure 53.

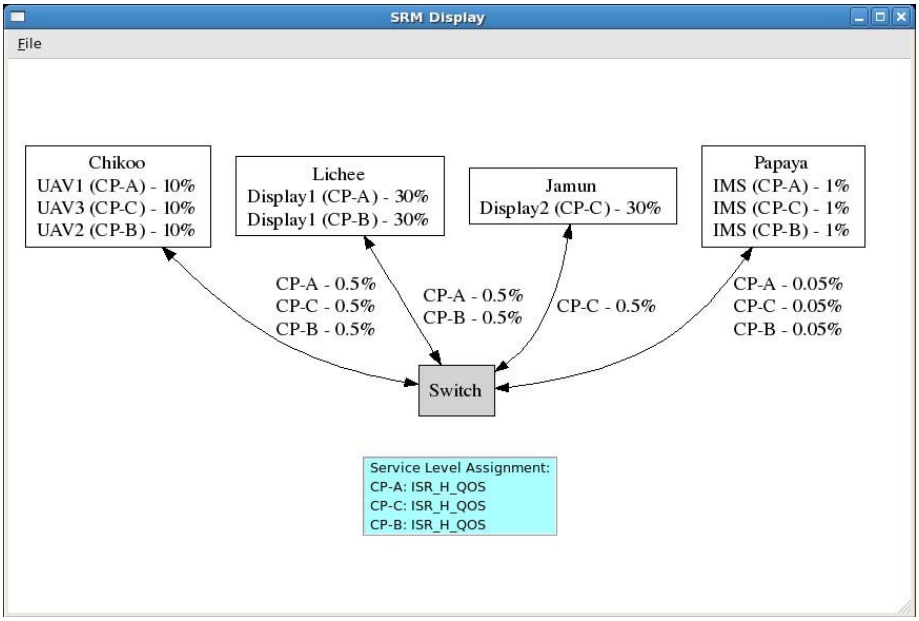


Figure 53: The SRM display showing the topology of the demonstration

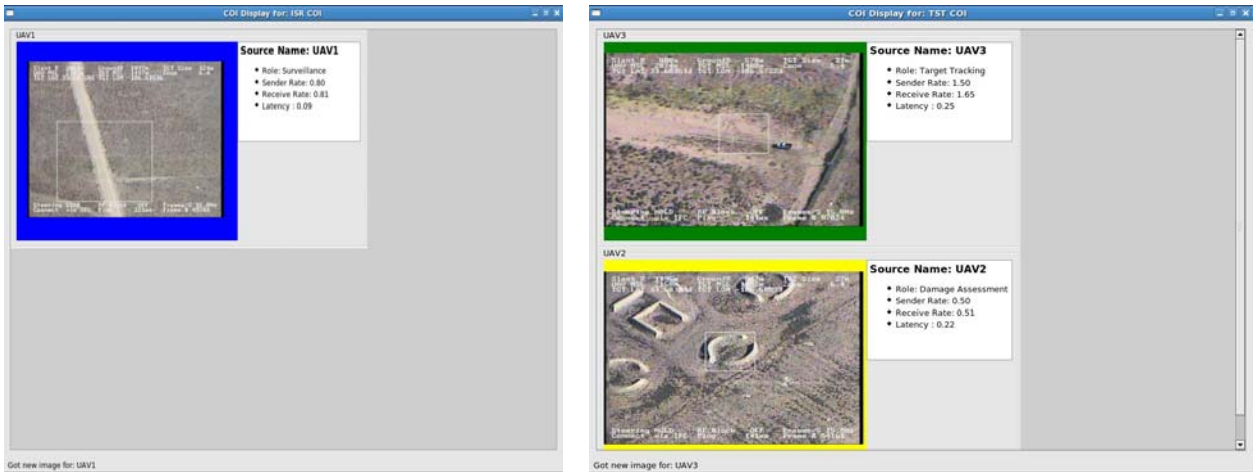


Figure 54: The C2 consumer client displays showing received imagery

The demonstration also included a display that showed the internal behavior of the demonstration software from different points of view, each on a different tab, including the following:

- The resource view showing each resource and how much of the resource is being used by each control point, illustrated in Figure 55.

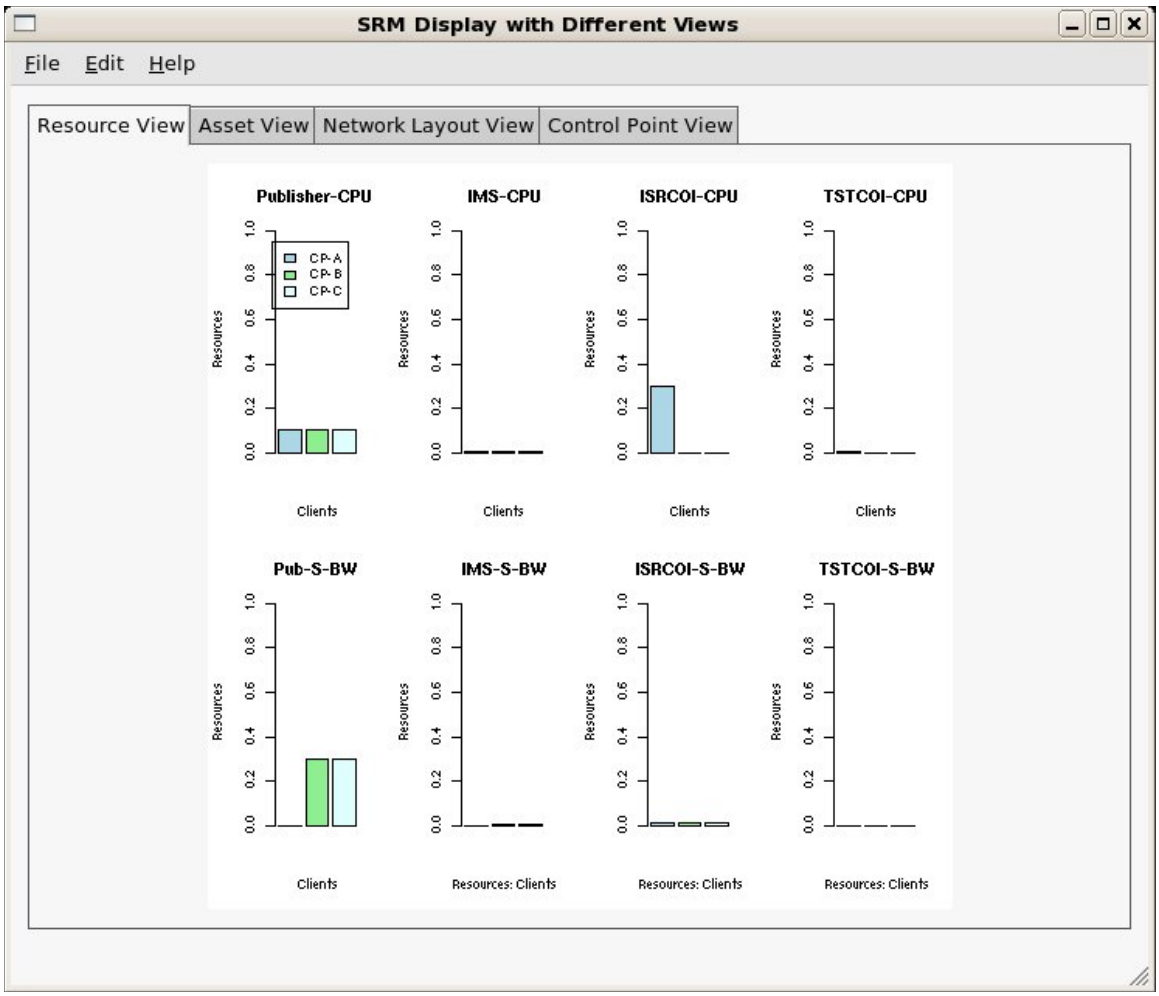


Figure 55: The resource view showing each resource and how it is allocated.

- The asset view showing the resource allocations for each control point, illustrated in Figure 56.

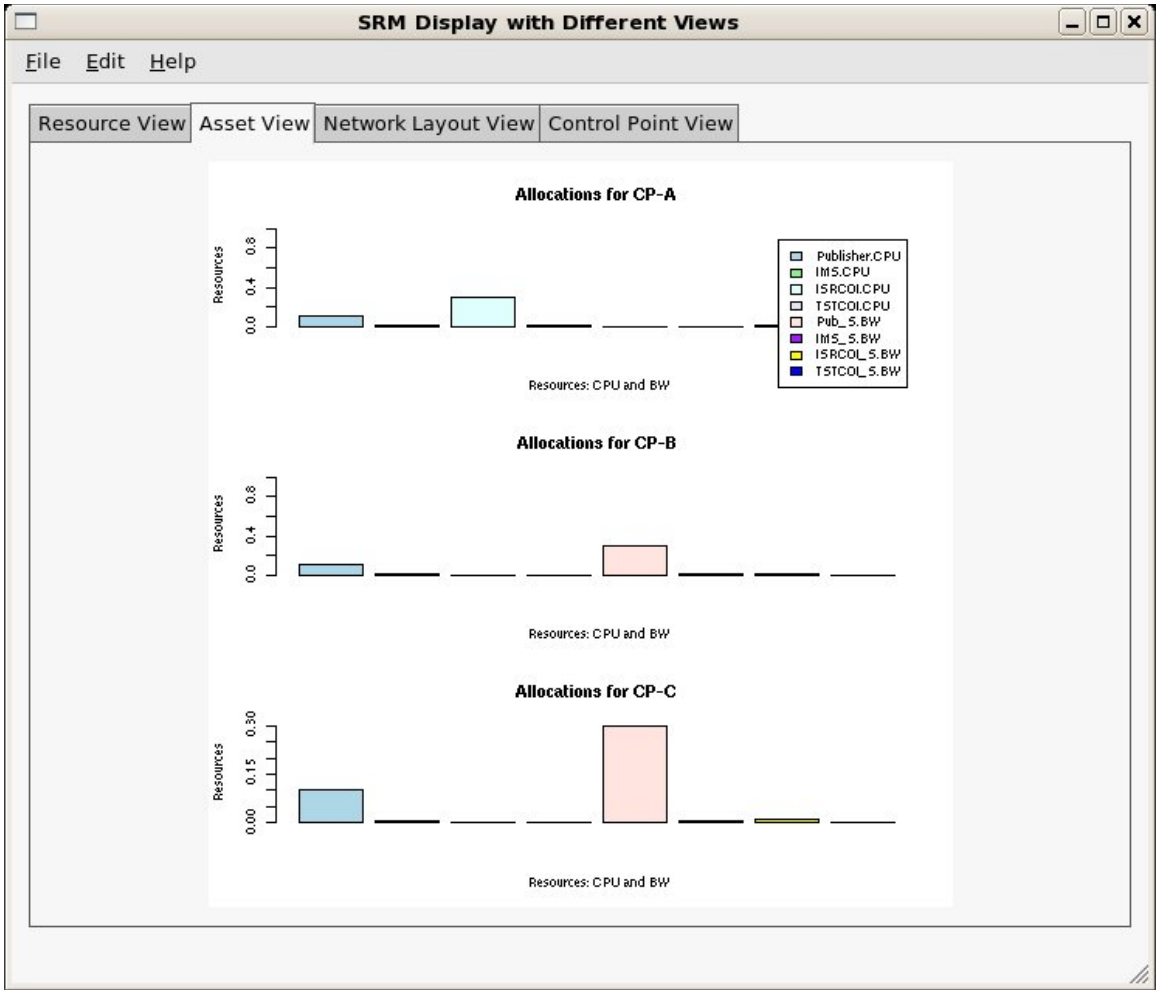


Figure 56: The asset view showing each control point and the resources allocated to it.

- The network layout view showing the topology and data flow of the demonstration, illustrated in Figure 57.

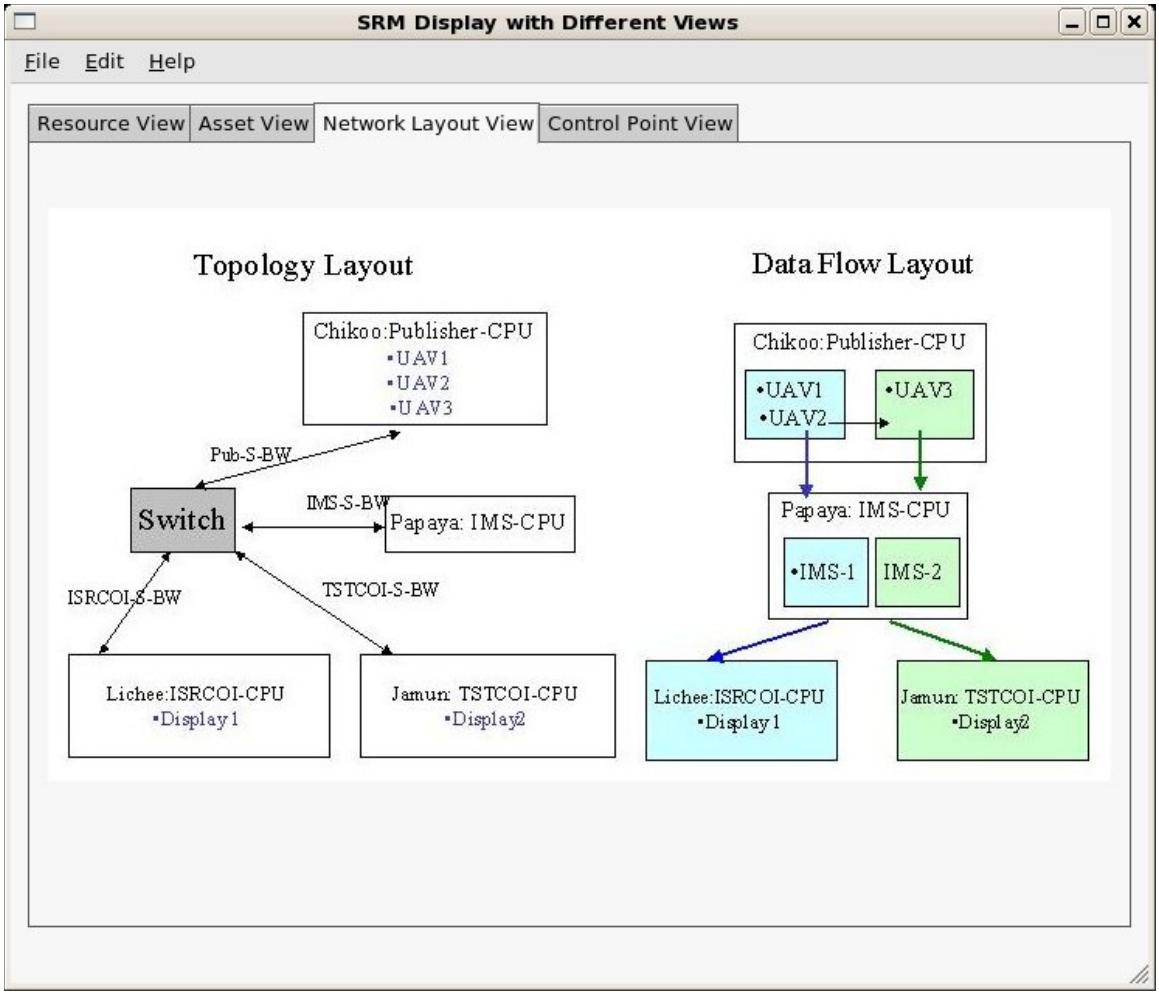


Figure 57: The network layout view showing the demonstration topology and data flow.

- The control point view showing which processes in the demonstration made up each control point, i.e., related elements that are treated as a single unit by the QoS manager, illustrated in Figure 58.

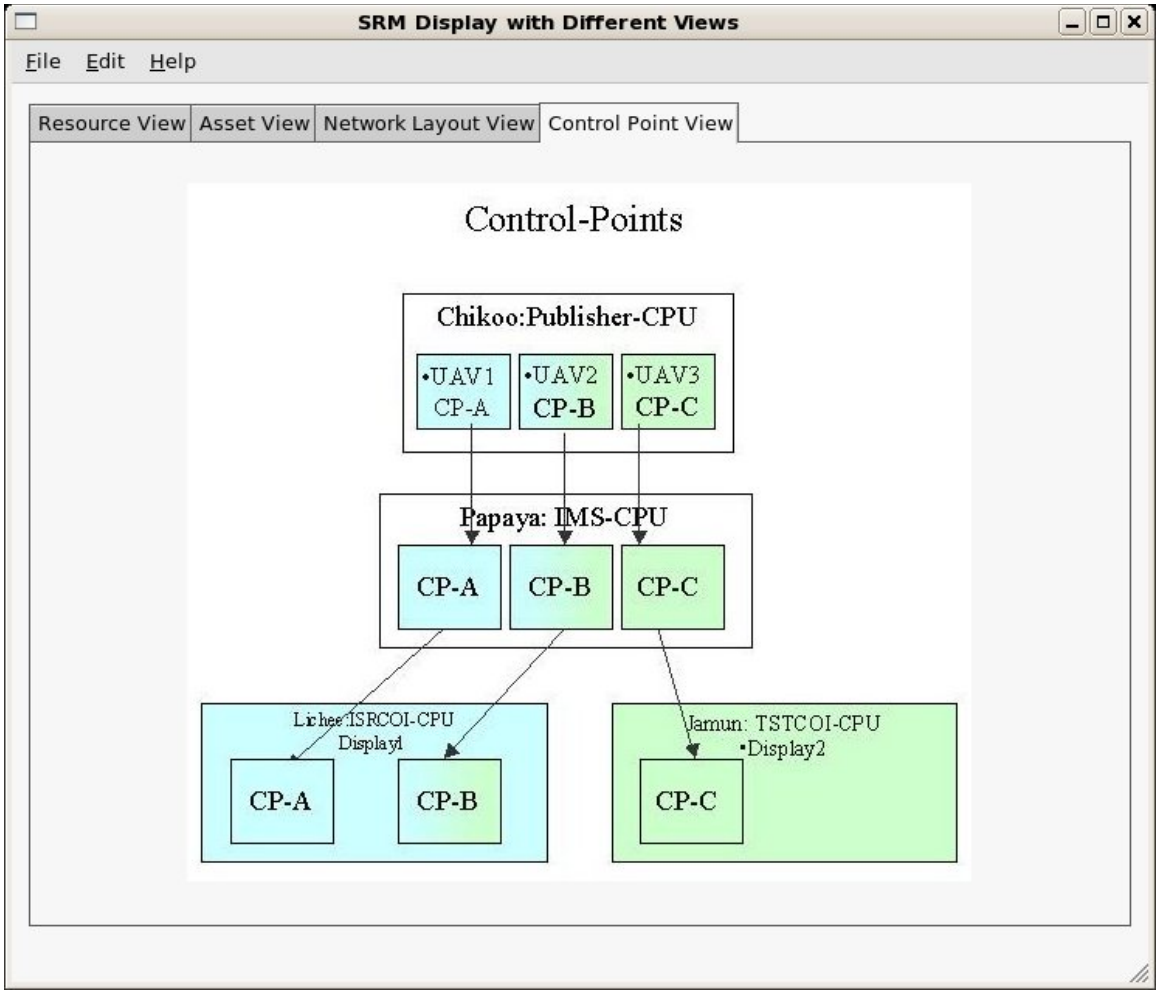


Figure 58: The control point view that shows the related elements in the demonstration that make up each control point.

7.3 Final Demonstration (Final TIM at AFRL, November 15, 2007)

At our final technical interchange meeting at AFRL on November 15, 2007, we conducted a demonstration of the multi-QoS, multi-resource enhanced SRM prototype described in Section 6. In contrast to the previous demonstrations, described in Sections 7.1 and 0, which concentrated on demonstrating QoS management in an operationally relevant context, this demonstration

concentrated on showcasing the SRM running the new QoS management algorithms, including its ability to allocate QoS levels for multiple information spaces with large numbers of applications. To do this, we needed to simulate many more applications than was possible in the previous demonstrations, which ran simulated clients with real IMSs and resources on a few laptops. This demonstration used configuration files, user-specified input, and a scenario generator to provide input for the SRM that simulated large numbers of applications, QoS levels, utility measures, and resource usage.

The demonstration, described in more detail in [2], showcased the extensibility and usability of the SRM, including the following:

1. *Configurability of the SRM with various allocation algorithms:* We provided the option of configuring the SRM with either an Optimizing Brute Force or Greedy Approximation second phase.
2. *Scalability to handle a large number of applications:* The demonstration illustrated that the SRM using the approximation first- and second-phase can provide allocations to a large number of applications and how the runtime is affected by the number of applications.
3. *Support for dynamic information spaces*: The demonstration illustrated that the SRM can allocate QoS levels in a dynamic information space environment where any of the following can change: the number of applications, the roles of applications, the number of resources shared across the information spaces, and the applications sharing resources across information spaces.

7.3.1 The Demonstration Context

The demonstration included two information spaces and a fixed number of 225 available resources. Each information space had access to 150 of the available resources, with 75 of the available resources available to be used by applications in either information space, as illustrated in Figure 59. We used a single instance of the SRM software to simulate the two SRMs normally used by two information spaces. Our single SRM ran the first phase algorithm, followed by the second phases for each information space sequentially, followed by the centralized Optimizing Brute Force algorithm (if the scenario was small enough) to get the optimal solution for comparison. The centralized Optimizing Brute Force algorithm would only be run if the number

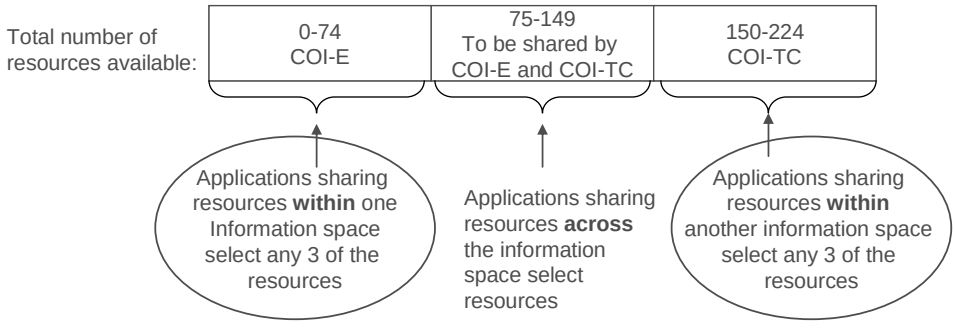


Figure 59: The final demonstration utilized 225 available resources, 75 of which were shared between the two information spaces.

of possible allocations (leaf nodes in the search tree) was less than or equal to 3^{50}, or 7.18E23, states based on our experimental runtime measures for the Optimizing Brute Force algorithm described in Section 5.3.1 and illustrated in Figure 18.

The number of applications running in each information space was entered at runtime using the Mission Manager interface shown in Figure 60. The QoS levels, utilities, and resource usage were provided in a configuration file, as illustrated in Figure 61. The exact resources that were used by each application and QoS level was mapped randomly by an instance of the Resource Mapper, described in Section 6.4.2, which did resource mapping randomly from among the available resources.

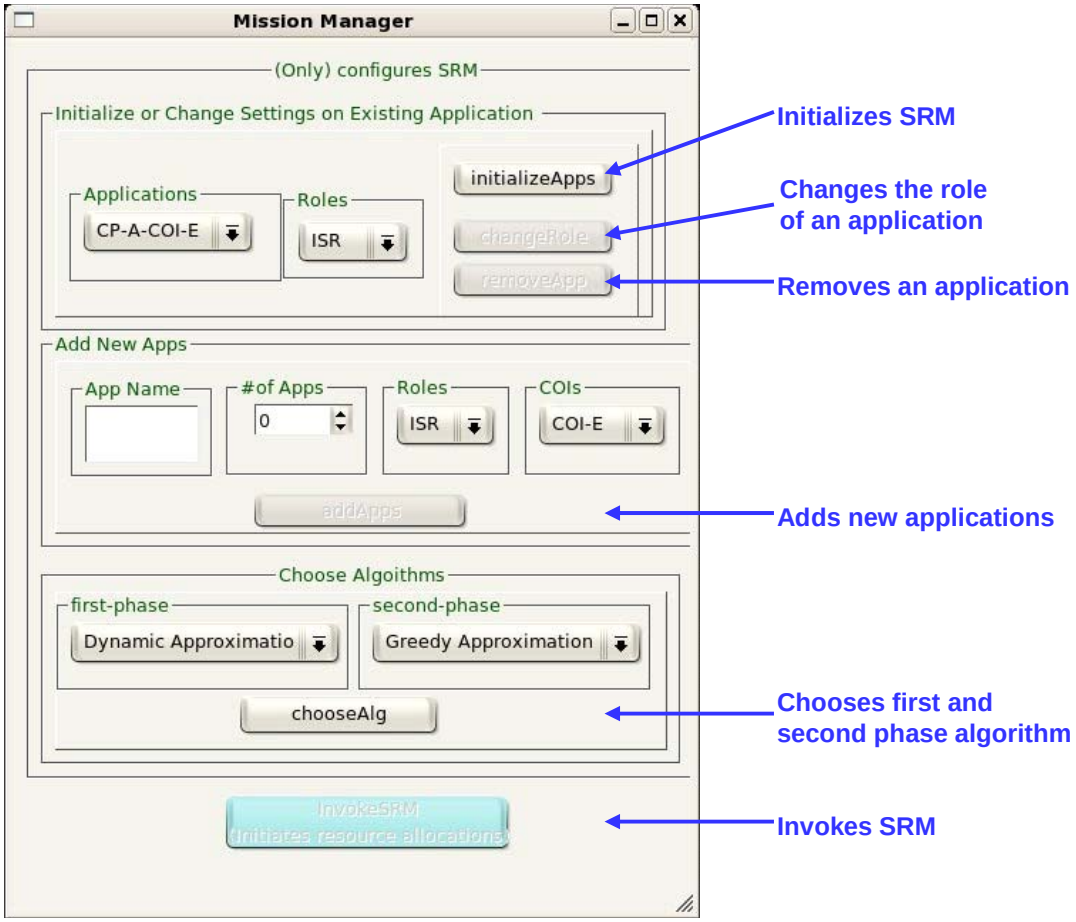


Figure 60: The Mission Manager interface used to drive the final demonstration.

Because the number of available resources was fixed, the contention for resources increased as we increased the number of applications during execution of the demonstration. The likelihood of resources being shared between the information spaces also increased, as did the number of applications sharing resources between the information spaces.

The two information spaces simulated in the demonstration were labeled COI-E and COI-TC. Each application could have two roles, ISR or TT. As illustrated in Figure 61, the configuration

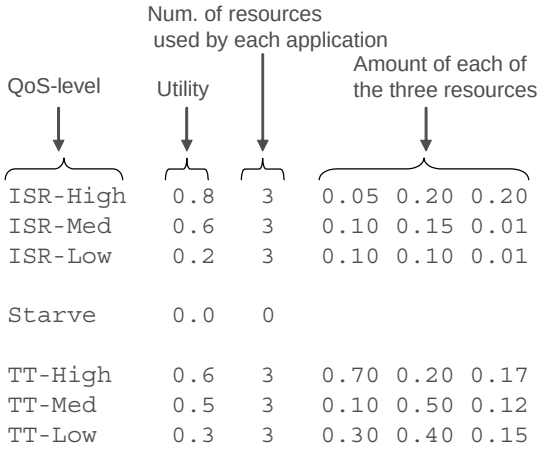


Figure 61: Example configuration file used in the final demonstration.

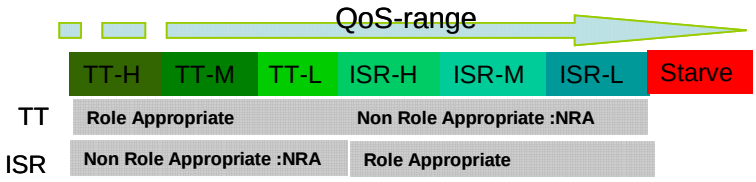


Figure 62: Three of the QoS levels were considered appropriate for a given role.

file used in the demonstration specified seven QoS levels: three for the ISR role, three for the TT role, and a starvation level. As illustrated in Figure 62, although any of the QoS levels could be chosen by the SRM, only three of them were appropriate for any application at a given time: the three pertaining to its role. To simulate this, the utility of each role-appropriate QoS level was multiplied by 1000 by the simulation software. This meant that the SRM calculated much greater utility for choosing a role appropriate QoS level, very small utility for choosing a non-role appropriate QoS level, and no utility for choosing starvation. For example, if an application is in the ISR role, the ISR utility values in the file were multiplied by 1000, making them 3 orders of magnitude higher value than the TT QoS levels.

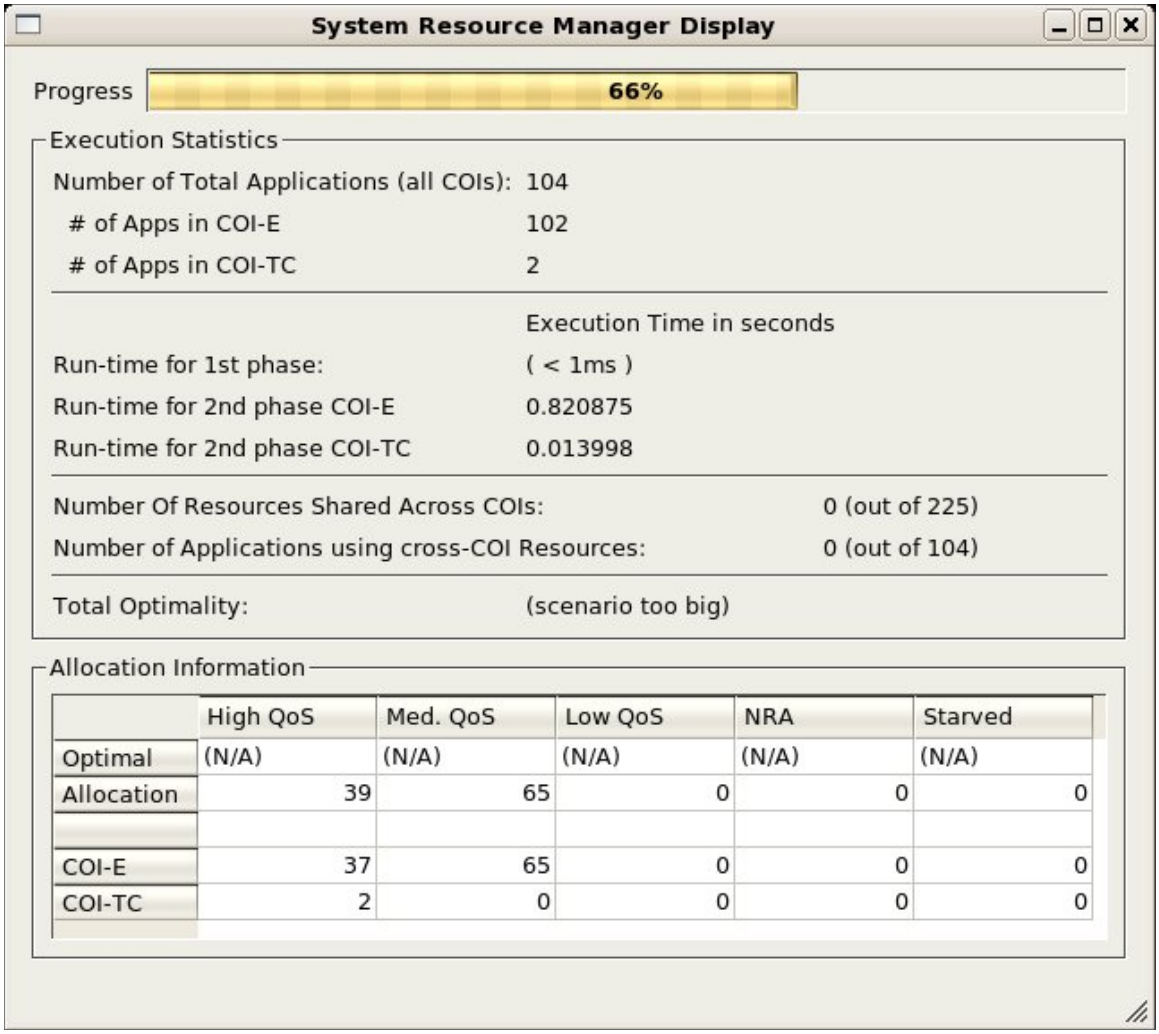
7.3.2 Execution of the Demonstration

We conducted the demonstration to show the SRM allocating QoS levels and resources as the number of applications varied, the roles of the applications varied, and the algorithms used by the SRM varied.

We started the demonstration by using the Mission Manager to initialize the SRM. This created the two simulated information spaces, COI-E and COI-TC, and placed two applications into each information space in predefined roles, specified by command line arguments to the

Mission Manager. Once the SRM was initialized with this start-up configuration, we took the following steps:

1. *Dynamically Configure SRM with a Pluggable QoS-Management Algorithm:* Using the Mission Manager interface we selected one of the second-phase algorithms. We were careful to select the Greedy Approximation algorithm when the number of applications in the information spaces got larger than a few tens of applications, because of the worst case execution time of the Optimizing Brute Force algorithm.
2. *Execution of the SRM*: We added applications, invoking the SRM afterward to show the QoS allocations, with the results displayed as illustrated in Figure 63. We started by adding a few (ten to twenty) applications to each of the information spaces and invoked the SRM. The results display showed the speed of execution of each phase and the resulting QoS allocation, i.e., how many applications got their highest QoS level, their next highest, their lowest, a non-role appropriate QoS level, or the



	High QoS	Med. QoS	Low QoS	NRA	Starved
Optimal	(N/A)	(N/A)	(N/A)	(N/A)	(N/A)
Allocation	39	65	0	0	0
COI-E	37	65	0	0	0
COI-TC	2	0	0	0	0

Figure 63: The display of the results of invoking the SRM during the final demonstration

starvation level (representing no allocation). At these small numbers, the results display also showed the optimal allocation (produced by running the centralized Optimizing Brute Force as a comparison).

3. *Scalability of the SRM:* We then added more applications incrementally to each information space, first 50-70, then hundreds, then a thousand or so, invoking the SRM after each addition to show the speed of the execution and the resulting allocation. As the number of applications rose, the execution time of the first phase began to dominate because the number of applications sharing resources between the information spaces became larger than the number of applications in either information space. Likewise, the resulting allocations had larger numbers of applications in the starved or non-role appropriate allocation categories. Both of these phenomena are because the number of available resources was fixed and as the number of applications increased, contention for this fixed pool of resources increased.
4. *Dynamic Environment:* We performed the following to show the dynamic nature of the SRM:
    a. We removed applications from each of the information spaces.
    b. We changed the role of some of the applications.

At each step, we showed the resource allocations for each COI, the time it took to run each of the first and second-phase algorithms, the number of resources that were shared across the information spaces, the number of applications that were shared across the information spaces, and the number of applications that got each possible QoS allocation. We also spent time during the demonstration allowing the observers to suggest configurations and inputs to try out.

8 Conclusions and Future Work Recommendations

The DynRIIC project has made advancements in the following two key areas for dynamic QoS management for publish-subscribe information spaces: architecture and algorithms.

We have defined an architecture for a multi-layered QoS management system, and produced a design and prototype of the architecture. The QMS architecture, design, and prototype extend the concept of information spaces to make them suitable for the real-time information delivery needed by tactical, asymmetric warfighting missions. The current QMS design fits well within the information space concept, providing a manager that works in conjunction with existing managers, such as IMSs, security and mission managers. The multi-layered architecture fits the hierarchical structure of military missions, while retaining the flexibility for distributed, cooperating missions.

We have also advanced the state of the art in multi-resource QoS allocation algorithms by defining, evaluating, and prototyping a set of algorithms that allocate QoS levels and resources across large numbers of applications and control points within information spaces, and for applications that share resources across information spaces. The Optimizing Brute Force algorithm provides optimal allocations in reasonable execution time for modest numbers of applications. The Greedy Approximation algorithm provides approximate solutions, but scales well, with demonstrated fast execution times to hundreds or thousands of applications. Greedy Approximation has the fastest runtime, but less optimal solutions, in highly contentious scenarios (defined by the number of feasible allocations). Conversely, it produces more optimal solutions, but takes more time to do so, when contention is low (i.e., there are many feasible solutions).

Our multi-phase approach to inter-information space QoS allocations scales well when the number of applications are spread relatively uniformly over the information spaces and when only a reasonably small number of the applications share resources across the information spaces. When the majority of applications share resources across information spaces or the applications are not uniformly spread, then the execution of the first phase or the second phase, respectively, dominate the execution time and approach the execution time of a centralized algorithm.

The architecture and algorithm results that we have produced in DynRIIC help advance the AFRL OIM goals of tactical information dominance (TID), as illustrated in Figure 64. Future work building upon these foundations will help further move toward the TID vision. Some of the areas that need future investigation follow:

QoS Policy Definition and Development. There are still important areas of research to investigate languages for QoS policy, aggregate policies for starvation and fairness, other methods for creating policies such as negotiation or commerce-based, richer QoS policy specifications for a larger class of systems and scenarios, and capturing mission-level policy and refining it to actionable, enforceable policies.

Cooperating, peer-to-peer QoS managers. Our multi-layered approach maps well to the hierarchical military structure. In the tactical environment, there are more peer-to-peer interactions. It is an important area of research to combine our approach with technologies for MANETs and ad hoc, peer-to-peer environments.

Synchronizing Information Spaces. Dissemination and enforcement of QoS policies must be carefully synchronized. The QoS policies must be received and acted upon by all the relevant

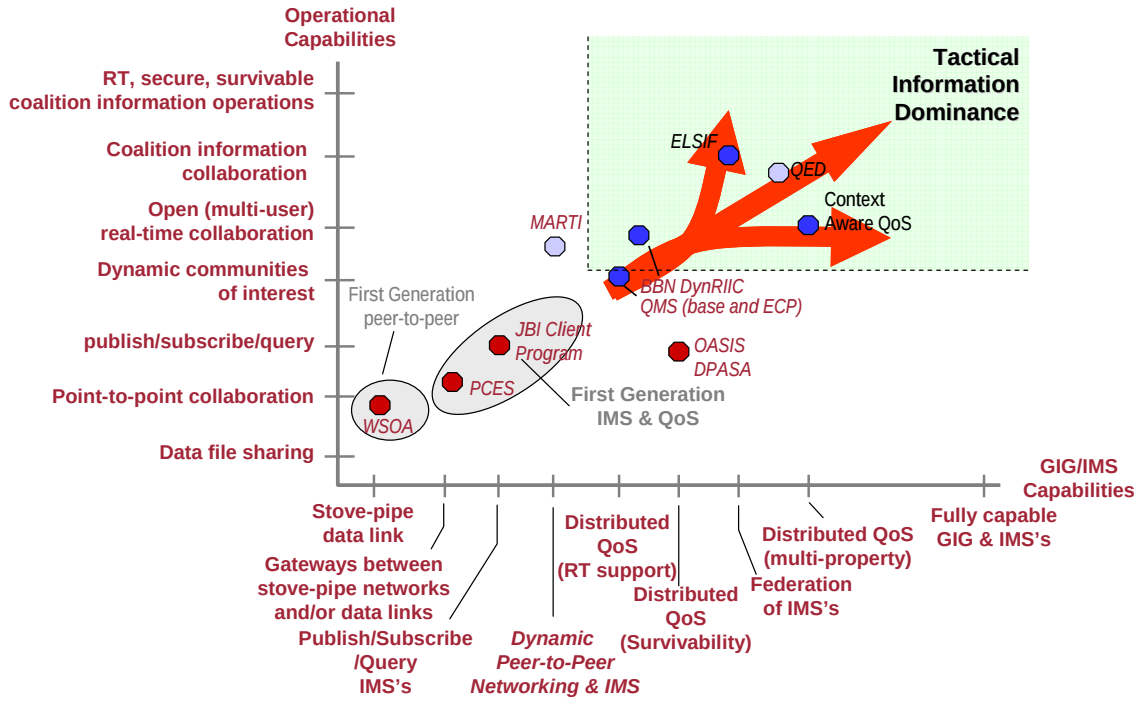


Figure 64: The DynRIIC project has advanced the goals of tactical information dominance and helped establish a foundation for further advancements toward the TID vision.

local enforcement points. Otherwise, the system could be running in an unstable state in which some participants are operating on current policies and others on previous ones. It is also important that applications do not start using resources before they are relinquished by other applications, otherwise an overload situation could occur. We have defined and prototyped some concepts contributing to this synchronization, but there is room for further research.

Acquiring Inputs to the QoS Allocation Algorithms. Our QMS algorithms require knowing the amount of available resources; a set of discrete QoS levels; the number, types, and amount of resources used by a given QoS level for an application; and the utility associated with an application and QoS level. Having good sources for the availability of resources and accurate estimates of resource use for a given application and QoS level is non-trivial. This information is essential for the QMS algorithms to allocate QoS levels and resources accurately. We need to provide or use sensors, monitors, and models that provide this information, and do so in a coordinated manner so that the algorithms in different information spaces see the same global view of available and required resources. This problem becomes even more difficult when the information is needed in real-time (so that important applications do not get starved), or when information spaces operate in disconnected and tactical environments (where the communications to the QMS can be unreliable).

Identifying Outliers. The QMS approximation algorithms produced some outliers (i.e., suboptimal solutions) for the randomly generated scenarios. Although we identified extreme resource contention as a cause for some of the outliers, it is still an open issue to identify the cause for others.

Predicting Optimality. Even if the source of many of the outliers could be determined, it is likely that not all of the outliers could be eliminated. That is, the approximation algorithm will perform better on some scenarios than others, in terms of producing near optimal solutions. Since we have a choice of QoS allocation algorithms, it is a desirable goal to find a metric, or set of metrics, that can be used to predict how well the approximation algorithm will perform for a given scenario.

9 Chronological Review of DynRIIC Activities

This section presents a description of the technical progress on the DynRIIC project, presented in chronological order, as reported in the DynRIIC monthly reports.

9.1 Project inception (September 22, 2005) through October 31, 2005

We kicked off this project this month, with a kickoff meeting held at AFRL, Rome, NY, on October 24. The slides from this kickoff meeting have been uploaded to Jiffy.

We began identifying example Communities of Interest (COIs) upon which to focus, began identifying different implementations of IMS to use for these COIs, began identifying the interfaces that need to be defined, and began planning the elements of the proof of concept demonstration for this project.

We registered in AFRL's CMDB and requested a copy of the AFRL RI 1.2.5 (patched) release and fuselet related software. As soon as we receive this software, we will begin evaluating it.

9.2 November 2005

During the report period, we identified a draft set of representative COIs extracted from COI reports we obtained from the Government and a literature search. We identified ways in which these COIs can interoperate (e.g., assets leaving, sharing assets, assets joining) and described these interoperations in concrete terms using scenarios from our previous demonstration for the DARPA/IXO PCES program.

We also developed a draft list of requirements and specifications for interfaces supporting interoperation between COIs from a functional and QoS management point of view.

We began developing a draft *User's Manual and Demonstration Script* that documents our demonstration context, representative set of COIs, set of potential interoperations, interface specifications, and demonstration architecture and design. We plan to deliver this document to AFRL during the next report period.

We received a copy of the AFRL RI 1.2.5 (patched) release, but have not yet received the copy of the fuselet related software that we requested.

9.3 December 2005

During the report period, building upon the draft requirements and representative COIs that we developed last report period, we developed a strawman architecture for a *QoS Management System (QMS)*. The QMS will handle the QoS management aspects of COI interoperation, including assets entering, leaving, and being shared between COIs. Once an entering asset has been assigned a role in a COI (e.g., a role in a mission such as surveillance or target tracking), the asset negotiates a *QoS contract* with the QMS, including what resources it provides, what resources it is allocated, and the necessary QoS tradeoffs it needs to make. The QMS includes a layered resource manager, with a *System Resource Manager (SRM)* that allocates the resources shared within a COI or COA and provides QoS policies and *Local Resource Managers (LRMs)* for each asset that shape the asset's behavior to effectively utilize the resources it is allocated in

service of its role in the mission. This architecture will be documented in our TIM presentation and in the next draft of the *User's Guide and Demonstration Script* document.

We delivered the first draft of the *User's Guide and Demonstration Script* that documents our demonstration context, representative set of COIs, set of potential interoperations, interface specifications, and demonstration architecture and design. We began developing our presentation for the upcoming TIM.

9.4 January 2006

During the report period, we hosted a Technical Interchange Meeting (TIM) at BBN. During the TIM, we presented the goals of our project, our architecture for a QoS Management System (QMS) supporting interoperability between COIs, and our plans. We received feedback on our architecture to date from Mr. Dale Richards.

After the TIM, we made progress in the following activities:

- We revised our architecture based upon the feedback we received from Mr. Richards.
- We began revising the User's Guide and Demonstration Script, based on the work we've done since submitting the first draft and based upon feedback from Mr. Richards, including updating the format and updating the system requirements.
- We began refining our demonstration scenario and identifying the constituent software pieces of the demonstration.
- We began designing the software elements of the QMS.
- We researched different IMS systems implementations – AFRL RI, Web Services, and DDS that we are going to support and use to test QMS. These are documented in the revised User's Guide and Demonstration Script.
- We started defining the QoS terminology such as QoS Policies and QoS Contracts and providing examples.

In addition, we contacted Boeing, another ICED contractor. We had identified the need for an element playing a *Mission Manager* role, granting permission to join (i.e., controlling access to) a COI and assigning roles in a mission. There is overlap with some of the work Boeing is doing. We plan to continue to explore this with them.

9.5 February 2006

During the report period, we primarily worked on revising the software architecture, design, and demonstration scenario and script. This is captured in the revised version of the *User's Guide and Demonstration Script*, which we submitted to Jiffy on February 23.

The main changes included in this document are:

- We revised our architecture to concentrate on the QoS management capabilities.
- We updated the format and updated the system requirements.
- We refined our demonstration scenario and identified the constituent software pieces of the demonstration.

- We defined the terminology and provided more examples.
- We identified the IMS implementations that we plan to support and use to test QMS.

9.6 March 2006

Our main technical accomplishments during the report period involved refining our design and beginning the development of our QMS system and demonstration.

Our refined design is documented in our slides from our TIM on March 29. Some of the refinements that we made are to identify specifically the structure of the membership contract that an asset publishes upon entry to a COI and the QoS policy structure used to disseminate policy between an SRM and LRMs. We also further refined the interactions in the system when an asset enters, leaves, when a reconfiguration occurs, and when QoS managed information exchange occurs.

We also began implementing our QMS system and our demonstration for the JBI PI meeting. We implemented a System Resource Manager (SRM) and Local Resource Managers (LRM). We adapted a set of simulated ISR sensors representing UAVs to send imagery over the AFRL JBI Reference Implementation version 1.2.6. We integrated our SRM and LRMs to control the QoS of image publishing based on the role of the UAV.

We also began developing our presentation, poster, and quad chart for the JBI PI meeting, scheduled for April 11-12, 2006.

9.7 April 2006

During the report period, we continued development of the QoS management system (QMS) for COIs. Some of the specific accomplishments in the QMS development are

- Developed and tested the membership contracts that a platform (i.e., a data producer or consumer) entering a COI publishes to introduce itself to the IMS. Membership contracts are based on the Force Templates concept.
- Created a prototype Mission Manager GUI to let a commander add assets to a COI, remove assets from a COI, move assets between COIs, assign mission roles, and QoS constraints to assets.
- Developed and tested QoS policy contracts that mission and resource managers (such as the system resource manager, SRM) use to disseminate QoS policies. We are in the process of updating the Mission Manager and SRM to use these XML-based contracts.
- Enabled the system to connect to another IMS running on a different OS (Windows XP) when a new COI is dynamically created.

We developed a demonstration system for the JBI PI meeting. The demonstration illustrated a set of simulated ISR assets and two COIs, a surveillance COI and a time sensitive targeting task force COI. During the demonstration, assets would move from one COI to the other and their roles would change (between ISR, target tracking, and battle damage indication). The QMS

system dynamically adjusts the resources and application behavior to support the requirements of the roles.

In preparation for the PI meeting, we ran into some challenges installing the JBI RI 1.2.6 on Fedora, the current version of Red Hat Linux. However, with some help from Tom Clark and Robert Grant to create new scripts to run the JBI RI specifically on Fedora, and some other configuration and version changes, we were able to get this to run successfully.

We updated the demonstration code to integrate the membership contract. An entering simulated asset publishes a membership contract and the System Resource Manager (SRM) subscribes to it, updates its list of known assets to which it allocates resources, and reconfigures if needed.

We attended the JBI PI meeting and presented the progress of our project and delivered a quad chart. We also presented our demonstration and poster at the demonstration/poster session on the evening of April 11.

We continued development of our software going into and after the PI meeting, making the improvements documented in Section 5.

We also hosted a visit by Boeing at BBN on April 29, in which we discussed synergy between our ICED projects.

9.8 May 2006

During the report period, we continued development of the QoS management system (QMS) for COIs. Some of the specific accomplishments in the QMS development are

- Implemented interfaces and adapter code that enable assets to publish XML-based membership contracts to the JBI RI, thereby enabling them to enter a QMS-managed COI dynamically.
- Implemented interfaces and adapters to enable the publishing of QoS policies, QoS policy templates, mission weights, and available and allocated resources to the JBI.
- Created a script for easier startup of the software.
- Added code to enable the deployment of new SRMs, LRMs, and QoS mechanisms dynamically using the mission manager GUI.
- Enhanced the mission manager GUI to enable assets to be added and to deploy the code simulating the assets dynamically.
- Enhanced the SRM resource allocation algorithm to enable it to handle dynamic numbers of assets under management and to better handle dynamic numbers of participants and shared resources. We will continue to refine this algorithm in the coming months.
- Wrote, tested, and integrated the code to monitor resource usage. This is an enabling capability for the Connectivity Monitor, which we are continuing to develop.

We began planning the upcoming TIM at BBN in June, including planning for a demonstration and planning the slide presentation.

We updated the User's Guide and Demonstration Script document and submitted it.

9.9 June 2006

During the report period, we hosted a Technical Interchange Meeting (TIM) at BBN. During the TIM, we presented the status of our project, the goals for the next three months, and ideas for future work based on the achievements of this project. We also presented a demonstration of the current version of our software.

We also continued development of the QoS management system (QMS), demonstration system, and user's guide document. Among the specific improvements to the QMS software during the report period are:

- We validated all the schemas against xsd 2.1.1 and Altova XML Spy 2006. We also validated all the xml against the schemas. This was to replace our parsers with the xsd-generated parsers, so as to increase the robustness of the system.
- We integrated the XML-based membership contract, QoS policies, total resources and weight information into the system.
- We developed the code needed to pass the membership contracts, QoS policies, total resources and weights XML information through the JBI RI. However, we ran into issues with the lack of real-time support to deliver this policy information (described in Section 9) reliably. Currently we offer alternatives for delivery of this policy information through separate channels, Notification Service, or the RI.
- Developed code to dynamically deploy SRMs.
- Developed the interfaces for the Connectivity Monitor and enhanced the code we have for monitoring and displaying usage information.
- Integrated OCI's OpenDDS into a subset of our demonstration.
- Began investigating PrismTech's DDS.

We continued the interaction with Boeing that we reported in the April report to identify synergies between our projects and develop a complementary vision for our activities. We have begun writing a white paper that lays out a Tactical Information Dominance vision and a set of next step ideas for enabling it.

9.10 July 2006

During the report period, we made significant progress in development of the QMS software and a demonstration depicting the functionality of the QMS, and started packaging the software for release.

Specific accomplishments for this period include:

- Finished integration of OCI DDS into our QMS software and demonstration. DDS defines more QoS controls than the JBI RI that we were utilizing previously.

However, the OCI implementation of DDS only implements three of the DDS controls: history, liveliness and reliability. These are not sufficient to get real-time behavior. For example, those parameters are not sufficient to achieve differentiated delivery of messages based on priorities.

- Completed the development and testing of a layered SRM that allocates shared resources to a set of coordinating COIs. We also made corresponding changes to the Mission Manager to control and demonstrate this feature of the QMS.
- Developed edge components as reusable parts of the Connectivity Monitor. These edge components are self-contained code that can be assembled into a system to monitor resource usage.
- Migrated communication of all the control messages to use the CORBA Notification service. These include control messages between the Mission Manager and an SRM and between assets and an SRM. The CORBA Notification Service provides more reliable delivery of control messages than the JBI RI because it uses different threads/channels for each communication path and introduces lower delay.
- Finished scripts that automate the build and part of the run process of the QMS demonstration.
- Finished development of the code and scripts that simplify the use of the QMS software. These include utilities that will add all the required schemas to the JBI in one step via a script rather than in multiple steps manually.
- Continued the investigation of PrismTech's DDS, which we hope will provide support for more of the defined DDS QoS controls.

We finished the first version of the joint white paper that we are writing with Boeing and that we mentioned in the last report. We provided it to a set of interested AFRL personnel during the report period.

9.11 August 2006

During the report period, we developed several features of the QMS software and the QMS demonstration system. We worked on packaging the software for release with an eye on an interim delivery in September. Finally we worked on the documentation in the Users' Guide and Documentation Script.

Finished investigation of PrismTech's OpenSplice DDS and RTI's DDS (formerly called NDDS). The QMS software and demonstration already utilize OCI's DDS, which is limited in its support for QoS management (implementing only three of the 22 QoS parameters, called *QoS policies* in the specification, defined by the DDS specification), an open-source version of DDS. We obtained evaluation copies of OpenSplice and RTI DDS to examine the support that they provide for QoS and how it compares to OCI's DDS. Both OpenSplice and RTI's DDS implement more of the QoS policies in the DDS specification, although not all and not to the same degree. The three DDS implementations differ in the OSes, middleware, and network protocol that they support, with some overlap. OpenDDS and OpenSplice both use TAO CORBA, although not the same version (OCI maintains their own TAO release). They also support different versions of the Redhat Linux operating system. RTI DDS uses its own

proprietary middleware and language to interface. It also uses UDP, thereby limiting the size of information to 64K or less in each packet. Because of the expense and time remaining in our current project (at that time), we did not integrate either of the other DDS implementations, although this is a possibility for future work.

We enhanced the SRM algorithm to include the expected and actual usage of resources in its resource allocation calculations. Without including these factors, the SRM could sometimes allocate excess resources to participants in resource rich situations, even when the QoS policy (or other factors) prohibited them from fully utilizing them. While this is an acceptable resource allocation strategy, especially in situations where the number of participants are fixed, it can be suboptimal in a situation in which new assets arrive dynamically, since the arrival of any new participant requires that resources be taken from existing participants. If the existing participants are not utilizing the resources anyway, better to keep them in reserve (maintaining a *resource slack*) to allocate to new participants. The new version of the algorithm should result in better performance (fewer reallocations) with dynamic numbers of assets, but no loss in quality over the original algorithm. We updated the Asset Communicator interface and the membership schema and XML files to support the extended algorithm. The Connectivity Monitor component of the QMS sends the actual usage information to the SRM. Assets provide their expected usage information in their membership contract when they first join the QMS.

We developed the capability to dynamically turn QMS on and off. QMS can be turned off in two configurations:

- Enabling the simulated assets to send unmanaged data directly to the receivers at the C2 center.
- Blocking the simulated assets from publishing any information.

The first option is provided as a part of the Mission Manager GUI. The second option is available through a command line interface.

Added calculation of the latencies involved with processing information within the IMS or dissemination service (Information Object in case of AFRL's JBI RI and CORBA structures in case of OCI's DDS) to our usage collection and QoS display.

Updated the simulated UAV sender to read imagery data from files in addition to its current capability of generating the imagery data from in real-time MPEG2. This is useful for smaller scale (fewer machines) demonstrations, since it enables more simulated information providers to be hosted on a single machine.

Updated the QoS Display to include additional useful information. First, it indicates the type of IMS that a COI is utilizing for sending data. Second, it indicates when resource usage or information quality (size, rate, compression, etc.) violates allocations and QoS policies, i.e., the displayed values turn red if the actual usage of resources or QoS attributes violates the ranges provided by the SRM. Finally, the GUI now also depicts the rate at which information is published to and consumed from an IMS and the latency values associated with sending information through an IMS.

We updated the Users Guide and Demonstration Script to reflect the current state of the QMS and demonstration software. This is ongoing as we test and dry run the software and will continue into the next report period.

We modified the demonstration scenario to depict the resource usage in a non QoS managed environment. This is an enhancement that will enable us to demonstrate the benefits of QoS management as part of the planned demonstration next period, by selectively turning off QoS management and showing the behavior contrasted with QoS managed interactions.

9.12 September 2006

During the report period, we prepared the QMS software, demonstration software, and the Users' Guide and Demonstration script for an interim v 1.0 release. This preparation included

- Packaging the software using meaningful names for the directory hierarchies so that users can easily navigate the QMS software, demonstration software, and other third party software.
- Writing make files and scripts to facilitate ease of building and deploying the software using a single command. For example, the QMS software and all the third party software (excluding JBI) can be built by executing the *make* command. All the components of the software can be deployed using one script.
- Testing the software exhaustively to iron out as many bugs as possible.
- Burning DVDs for easily handing over the released software.
- Polishing (editing) the Users' Guide and Demonstration Script to simplify the text with a goal of ease of use of the document with the released QMS software and demonstration software.

Also, during the report period, we prepared for a TIM at Rome Laboratory, scheduled for 2 October.

9.13 October 2006

During the report period, we visited AFRL on 2 October and held a technical interchange meeting with Capt Marckenson Dieujuste, our Laboratory Program Manager. Other AFRL personnel, including Bob Hillman and James Hanna, also attended.

At the TIM, we presented our technical progress, demonstrated the current capabilities of the ICED QMS software, distributed DVDs containing the software, and distributed draft copies of the User's Guide and Demonstration Script documentation.

We attended the Systems & Information Interoperability Branch Workshop at the Minnowbrook Conference Center on October 24 through October 27. Joe Loyall led a breakout session on QoS Enabled Dissemination. Copies of the organizing materials were distributed to attendees. Copies of those materials and outbrief materials were delivered to AFRL at the meeting.

9.14 November 2006

During the report period, we identified alternative approaches to multi-SRM coordination. Approach possibilities include

- SRMs allocate for specific sets of resources and send allocations directly to the local resource managers of clients using those resources. This requires careful coordination of when LRMs respond to resource allocations and overlaying a transaction process or some other means to avoid race conditions.
- SRMs follow a strict hierarchy, whereby SRMs at higher levels (such as those associated with communities of action) push policy to SRMs at lower levels (such as those associated with communities of interest). This requires defining the nature of the SRM to SRM policy.
- SRMs coordinate through negotiation. This requires an SRM discovery mechanism and negotiation protocol, as well as ensuring that the negotiation can be conducted in-line and meet real-time requirements.

We continued to work with AFRL Vanderbilt, Boeing, and IHMC to put together a draft project plan for QoS enabled dissemination based on the breakout session at the Systems & Information Interoperability Branch Workshop at the Minnowbrook Conference Center on October 24 through October 27.

9.15 December 2006

During the report period, we continued to design an approach to multi-SRM coordination, concentrating on a strategy in which SRMs are arranged in a hierarchy and SRMs at higher levels (such as those associated with communities of action) push policy to SRMs at lower levels (such as those associated with communities of interest).

We also continued to define strategies for multi-resource provisioning, exploring a knapsack approach and a greedy approach, investigating their orders of complexity and relation to known optimization techniques, and searching the literature for viable approaches to this known-to-be-complex problem.

Working with AFRL, Vanderbilt, Boeing, and IHMC, we put together a first complete draft of a project plan for QoS enabled dissemination.

9.16 January 2007

During the report period, we investigated several algorithms for multi-resource allocation, including the following:

- A brute force algorithm that explores the full space of combinations of applications and QoS levels. This algorithm is guaranteed to produce the optimal (i.e., highest utility) choice of applications and QoS levels that will fit a particular availability of resources. However, in worst case it can take exponential time in the number of applications and QoS levels for each application. We added two optimizations that prune the search space significantly in many cases.
- An algorithm, based on a paper by Toyoda, which provides an approximate solution to multi-dimensional knapsack type zero-one programming problems.

- A dynamic programming algorithm.

We developed a simulator for comparing the algorithms, and prototyped the first two of the above algorithms. The simulator generates random sets of scenarios containing a number of applications, QoS levels, and resource usage. We are using the simulator to compare the approximate solutions of the second algorithm to the optimal solutions and the speed of the algorithms.

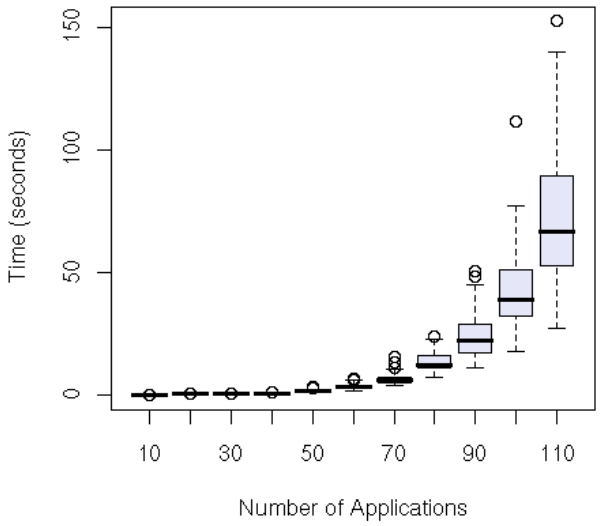
We began prototyping the third algorithm also and began identifying the characteristics of scenarios in which each algorithm performs more or less well.

We responded to feedback from AFRL on the draft project plan that we have put together for QoS enabled dissemination (along with Vanderbilt, Boeing, and IHMC).

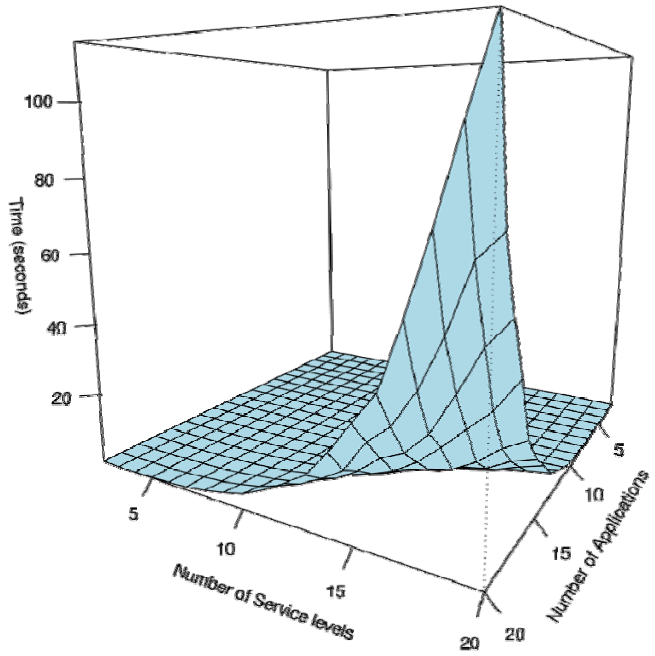
9.17 February 2007

During the report period, we continued our investigation of multi-resource allocation algorithms. In the last report period, we had prototyped a *brute force* algorithm that walks a tree of applications and service levels to find an *optimal* combination of applications and service levels. Optimality is defined by a combination of applications and service levels that has the highest utility among those that are feasible within the available resources. Since the brute force algorithm is exponential, $\Theta(q^a)$, we implemented two strategies that prune the search tree: first by pruning subtrees that are clearly not feasible and pruning subtrees that cannot lead to a higher utility than a feasible solution that has already been found.

During the current report period, we continued analyzing the Optimizing Brute Force and other algorithms. Analysis of the *Optimizing Brute Force* algorithm indicated that – on a set of randomly generated scenarios with a varying number of applications, but a fixed number (3) of service levels – the algorithm can return an optimal allocation within a few seconds up to approximately 50 applications. After that, the time increases exponentially, as shown in the following graph:

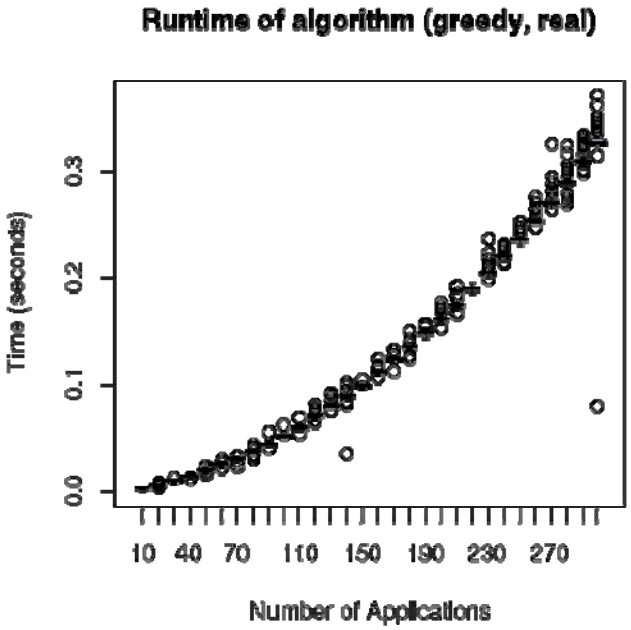


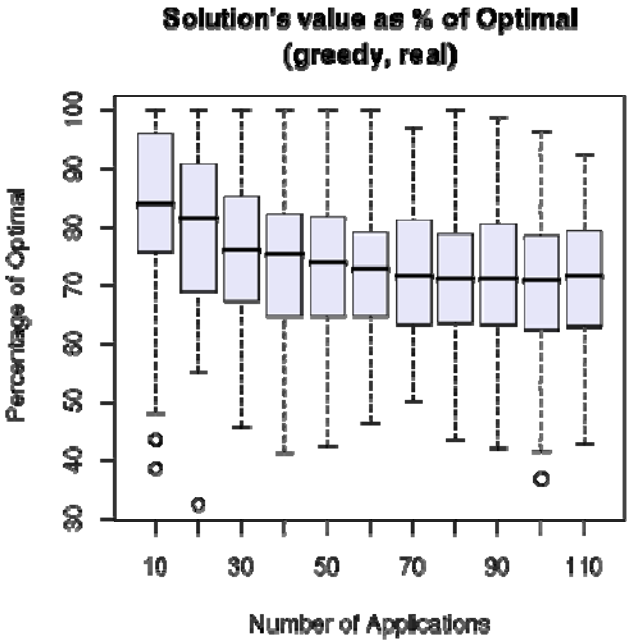
Varying both the number of applications and the number of service levels, as in the following graph, indicates that the performance is reasonable up to about 10 of each.



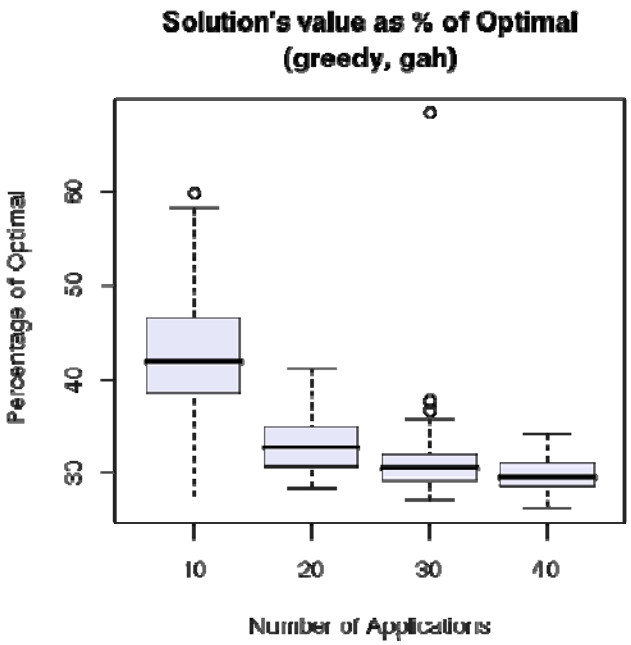
The Optimizing Brute Force algorithm is still exponential in the worst case, $O(q^a)$, but is significantly quicker in most scenarios. The non-optimizing brute force takes several hours to handle 20 applications with 3 service levels, whereas in 100 randomly generated scenarios of that size, the worst observed time for the Optimizing Brute Force was 45 milliseconds. Furthermore, the non-optimizing brute force algorithm has a much steeper execution time growth curve in the usual case. Going from 10 applications to 20 applications with the non-optimizing brute force goes from a runtime of approximately 250 ms to over 2 1/2 hours, approximately 36,000x. In contrast, going from 10 applications to 20 applications with the Optimizing Brute Force goes from 4 ms worst observed time to 45 ms, approximately 11x.

Also during the previous report period, we had implemented a Greedy Approximation algorithm, mapping the multiple resource allocation problem to a 0-1 integer programming problem and adapting an algorithm from a paper by Toyoda. During the current report period, we simulated the greedy algorithm on a set of randomly generated scenarios. The results, illustrated in the following two graphs, showed that the Greedy Approximation algorithm finds a solution much quicker ($O(a^2qr)$; up to 1200x faster for 110 applications) than Optimizing Brute Force, and the solution is a median of approximately 75% of optimal.

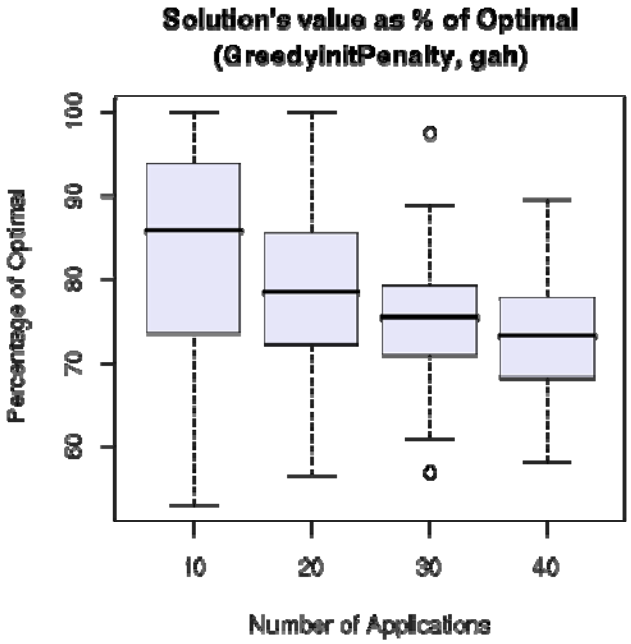




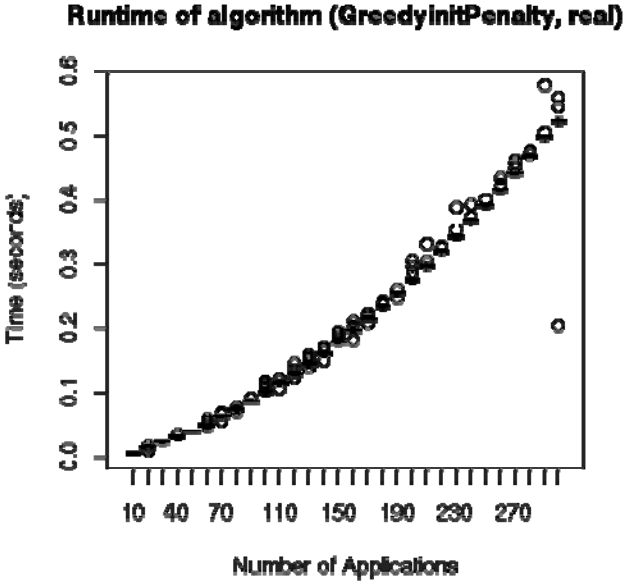
However, the analysis also pointed out a set of scenarios in which the Greedy Approximation algorithm performed particularly poorly. As illustrated in the following graph, for those scenarios (called *Greedy Achilles Heel* or *GAH* scenarios), the algorithm returned solutions that had a median of 30-40% of optimal:



The common attribute of those scenarios is that they included a single (or small number of) high priority applications that used most of a highly contended resource. The Greedy Approximation algorithm uses an *effective gradient* ratio of utility increase divided by resource usage and a *penalty vector* that increases the cost of using resources as they get scarce. We added an optimization that performed a quick analysis of how heavily resources are used by the applications and assigned initial values in the penalty vector to represent the projected contention for the resources. This optimization improved the performance of the algorithm on the GAH scenarios back to a median of 75-85% of optimal.



The runtime of the greedy algorithm is still $O(a^2qr)$, although shifted upward by a constant multiplier. The following graph shows that, on the same set of random scenarios as before, the greedy algorithm with initial penalty is approximately 600x faster than Optimizing Brute Force on 110 applications (versus 1200x for the version without an initial penalty).



Also during the report period, we implemented a third multi-resource allocation algorithm. This algorithm, called *Dynamic Programming for 0-1 Multi-Dimensional Multi-Constraint Knapsack Problem*, or simply *MMKP*, maps the problem to a 0-1 knapsack problem and adapts dynamic programming to solve it. Dynamic programming relies on breaking a problem into

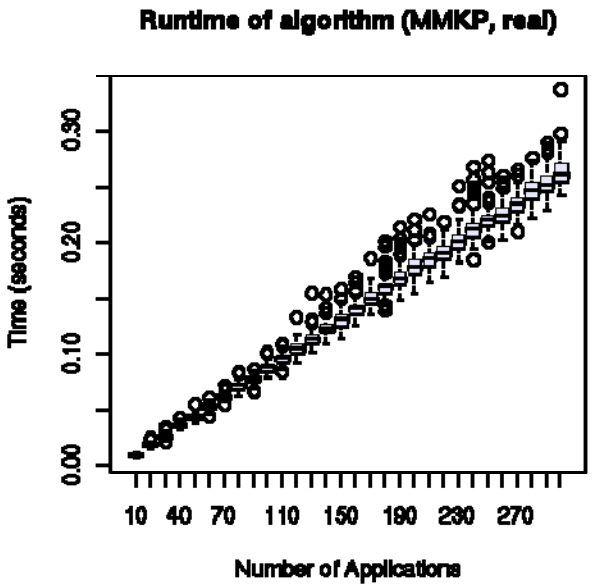
optimal subproblems, i.e., if solutions to the subproblems are optimal, then the solution to the main problem is optimal. In this case, we break the multi-resource allocation into subproblems as follows:

$$B[i, r] = \begin{cases} \text{if } r_i <= r & \max (B[i-1, r], B[i-1, r-r_i] + b_i) \\ \text{else} & B[i-1, r] \end{cases}$$

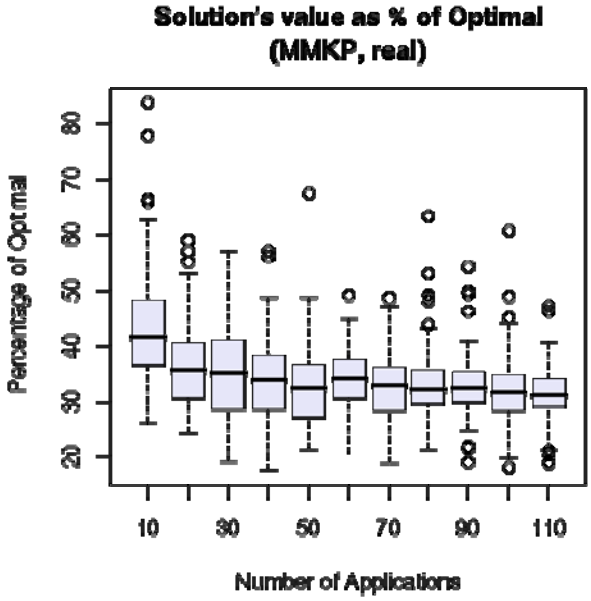
In English, the benefit (B[i,r]) of some point in the tree where there is the choice of deploying an application and service level pair (denoted *i*) that uses r_i amount of resources, is

- If there are enough resources ($r_i <= r$) to deploy the application and service level choice at *i*, then it is whichever of deploying *i* or not deploying *i* that results in the maximum benefit. Deploying *i* ($B[i-1, r-r_i] + b_i$) increases the utility by b_i but goes on to try to allocate the rest of the applications in the fewer resources, $r-r_i$. Not deploying *i* (*B[i-1, r]*) moves on in the tree to allocate the rest of the application-service level pairs with the available resources, *r*.
- If there aren't enough resources to deploy *i*, then it moves on to allocate the rest of the application-service level pairs with the available resources, *r*.

The runtime of the MMKP algorithm is *O(aqr)*, where *a* is the number of applications, *q* is the number of service levels, and *r* is the number of distinct resource levels. Varying only the number of applications, as before, the runtime grows approximately linearly.

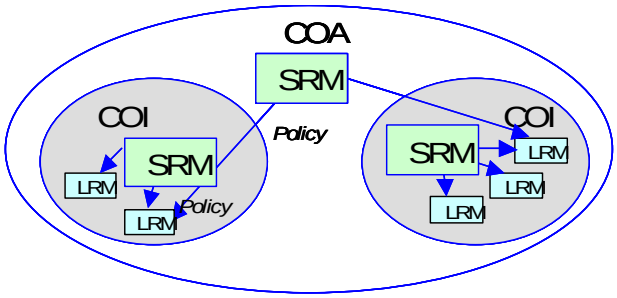


Quantization of the resources is important in this algorithm. The above graphs used a 0.1 quantization of resources, where resources are allocated in tenths. With this quantization, the same set of random scenarios as before provides solutions with a median between 35-40% of optimal.



A finer quantization could result in more optimal solutions, but would also result in a larger search space and slower time to reach a solution.

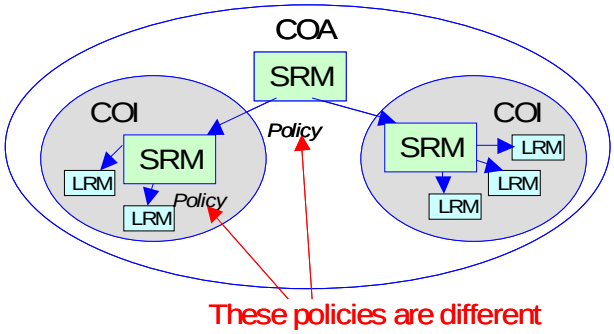
During the report period, we also investigated different ways in which SRMs could be layered to provide effective allocations and QoS policies. In our earlier work presented at the TIM on Oct 2, 2006 at AFRL, we had described and prototyped a *layered distributed approach* where there was one SRM per COI (SRM-COI) and one SRM per COA (SRM-COA). SRM-COI allocated resources and sent policies to the clients sharing resources within a COI; and SRM-COA allocated resources and sent policies to the clients sharing resources across COIs. Each of these SRMs communicated directly with the LRMs associated per client in the COIs.



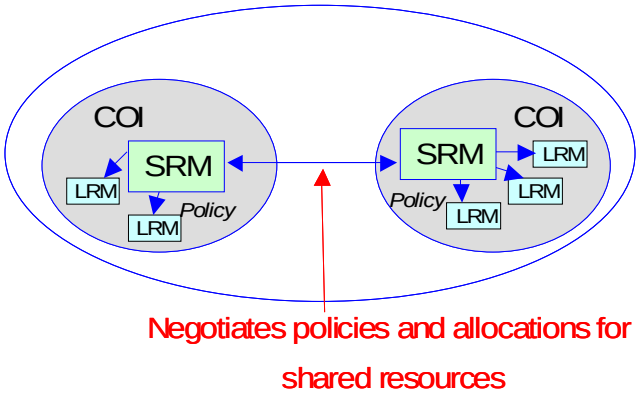
This approach has the advantages that it is scalable and all SRMs utilize the same algorithm. However, it has to be carefully managed or can lead to race conditions. An LRM could receive allocations and policies from all SRMs but not in the order in which they were sent. Alternatively, an LRM could get policy from one SRM and allocations from another or vice versa. If the LRM acted on every message, it could lead to thrashing or unintended behavior.

To overcome these issues, we investigated the following alternate approaches to cooperating SRMs:

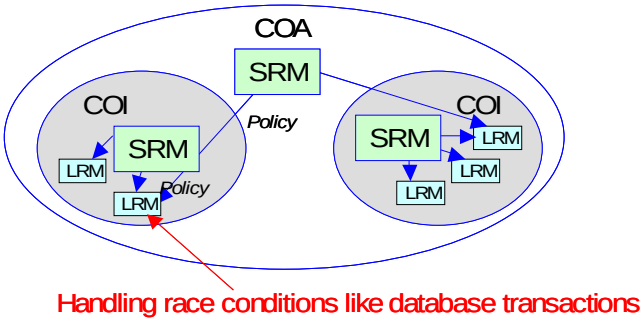
1. *Hierarchically Layered SRMs*. Under this approach, SRMs are hierarchically layered. SRMs for COAs construct policies, including allocations for resources shared between COIs, and sends the policies to COI level SRMs. COI SRMs then allocate resources and send policies and allocations to LRMs. Since only COI level SRMs direct LRMs, the possibility for race conditions is potentially eliminated. However, the SRMs at different levels are now different with different algorithms and the policy sent from a COA SRM to a COI SRM is different from that sent from a COI SRM to an LRM.



2. *Peer-to-Peer SRMs with Negotiations*. In this approach, SRMs reside only in COIs. SRMs negotiate policies and allocations with other SRMs for resources shared across COIs. The information exchanged in a negotiation needs to include the priorities of clients in a COI, the priority of a COI, and the resources that a client in a COI shares with other COIs. The benefit of this approach is that there are no supervisory SRMs. However, this approach could potentially require exchange of significant state information for negotiations; requires a discovery service for SRMs to discover other COIs with whom they are going to share resources; and needs a mediator or default policy in case negotiations do not converge.



3. *Layered Distributed SRM with Transactional Semantics*. This is an extension to our previous approach, which handles race conditions by providing transactional semantics on policy processing in LRMs, similar to transactions in databases. This approach would handle the possibility of race conditions, but adds complexity into the LRM and could adversely impact the performance of the system.



Our current plan is to prototype the hierarchically layered SRMs of approach 1.

Also during the current report period, we improved the simulator that we reported on last period, to add more options for scenario generation and simulation. The simulator and scenario generator were used for all the experiments described above.

We began planning and preparing for a technical interchange meeting at AFRL. The meeting will take place in early March and will be reported on in next month's report.

We held a telecon with AFRL and other QED planners (Vanderbilt, Boeing, and IHMC) to review the draft project plan that we have developed.

We also began working on our paper for the SPIE conference, Defense Transformation and Net-Centric Systems 2007, which we will be submitting next report period.

9.18 March 2007

During the report period, we began implementing the multi-resource allocation algorithms in the system resource manager (SRM) component. We created a shell for the SRM into which algorithms can be inserted and implemented the Greedy Approximation algorithm, described in last month's report.

We also began designing and developing a demonstration for the April PI meeting. The demonstration will show the multi-resource allocation capability and will include displays illustrating the effects of QoS management and the internal behaviors of the QoS management system.

During the report period, we visited AFRL on 2 March and held a technical interchange meeting with James Hanna and Robert Hillman. The presentation from this TIM was uploaded to Jiffy on 5 March.

Also during the report period, we began working on our presentation, poster, and quad chart for the PI meeting. We also finished writing our paper for the SPIE Defense Transformation and Net-Centric Systems 2007 [14]. We ran some additional experiments on the QMS software, graphed the results, and incorporated them into the paper.

We continued working on the QED plans. We held a telecon with Vanderbilt, Boeing, and IHMC, created a strawman architecture from that telecon, and planned face-to-face meetings at the SPIE conference and at the April PI meeting.

9.19 April 2007

During the report period, we continued our development of multi-resource allocation algorithms for the QoS management system. We implemented a version of the Greedy Approximation algorithm in the QMS's system resource manager. We also developed enhanced resource and behavior monitoring capabilities, and an enhanced set of displays to depict the application and QMS behavior in an information space. The enhanced allocation, management, and monitoring capabilities were showcased in the demonstration software that we developed for the PI meeting.

The enhanced displays depict the clients in an information space and the internals of the system resource manager (SRM). The client display shows which information space a publisher is publishing to and illustrates when a client moves between information spaces. The SRM display shows several views: a network, a resource and a client/application view. The network view shows the network topology, illustrates how data flows from a publisher to a consumer via the IMSs, and shows the number of resources needed and allocated along each path. The resource view depicts the total amount of each resource allocated across all clients. The client/application view depicts all the resources allocated to each client.

We also developed profiler software that measures the resource (CPU and bandwidth) use of an application. We used the profiler software to measure the resource usage of each client and of the IMS. The profiler collects CPU usage using the *rusage* system call and bandwidth by monitoring the *READ* and *WRITE* system calls made on INET sockets.

During the reported period, we also upgraded our software to use the current version of ACE/TAO (version 5.5.7/1.5.7) and the Linux operating system (Fedora Core 6).

We designed and developed a software demonstration showcasing the use of our enhanced system resource manager with multi-resource allocation and the new displays described above, in a dynamic information space scenario. The demonstration had two information spaces, the first with two publishers and one consumer and the second with one publisher and one consumer. The QMS allocated resources across both information spaces with multiple shared resources. During the demonstration, we changed the roles of clients and highlighted the corresponding reallocation of resources in response. We also moved clients from one information space to the other and again demonstrated the automated reallocation of resources in response. The demonstration illustrated managed predictable information delivery resulting from QMS control throughout the demonstration. To clearly illustrate the benefit of QMS management, we also demonstrated turning the QMS off and allowing the information spaces to run without the benefit of QMS control (with contrasting unpredictable and worse performance).

We attended the JBI PI meeting, presented the progress of our project, and delivered a quad chart. We also presented our demonstration and poster at the demonstration/poster session on the evening of April 11.

We continued our work on QED planning. We conducted two face-to-face meetings with Vanderbilt, Boeing, and IHMC at the SPIE conference and several conference calls at other times. We created a strawman architecture for the QED capability and led a QED planning meeting at the OIM PI meeting during which we presented and discussed the architecture.

9.20 May 2007

During the report period, we worked on developing and evaluating algorithms for managing resources that are shared across information spaces. In previous work, we developed algorithms that assign QoS levels to clients and QoS control points within information spaces, considering the multiple resources that might be used by those QoS levels. This new investigation considers that some of the resources used within an information space might also be simultaneously used by clients in other information spaces.

In order to build upon our multi-resource algorithms, our initial approach uses a two-phased algorithm. The first phase identifies the resources shared between information spaces and divides them between the information spaces. The second phase then runs the multiple-resource algorithms we developed earlier (e.g., the Greedy Approximation or Optimizing Brute Force) within each information space, with each information space only considering its allocation as the total available of the shared resources.

We have identified three first phase algorithms to evaluate:

1. *Even Splitter* – Each information space is given an even amount of each shared resource. For example, if two information spaces are sharing a resource X, each would be allocated half of X.
2. *Weighted Splitter* – Each information space is given a share weighted by a factor that can be provided by a commander or computed from the relative priorities of the information spaces.
3. *Dynamic Approximation* – The approximation algorithm is run on a subset of control points, i.e., those that share resources across the information spaces, to calculate the amount of each shared resource allocated to each information space.

The first two of these are simple approaches that serve two purposes. First, they are simple to implement, scalable, and likely to be sufficient for some situations. For example, an even split might be sufficient for information spaces with relatively uniform makeup and equal priorities. The weighted split might be sufficient for information spaces with significantly different priorities. Second, they serve as baselines against which to evaluate the third splitting algorithm, i.e., to see whether the additional dynamic processing to enable the third splitting provides significant improvement over the static splitting of the first two approaches and in what types of scenarios.

These algorithms can be run in a hierarchical or in a peer-to-peer distributed fashion. In the hierarchical configuration, a cross-information space QoS manager layered above the information space QoS managers would perform the first phase algorithm and provide the allocation computed as the total amount of resources available to each information space QoS manager. In the peer-to-peer configuration, each information space QoS manager runs algorithms for both phases. Since the first phase is run with the same inputs (i.e., the set of resources shared between information spaces and the control points and QoS levels using them), they each come up with the same results. They then run the second phase to calculate the QoS levels for control points within the information space, constrained by the amount of shared resources they allocated to themselves in the first phase.

As another set of baselines, we also prototyped two "global" algorithms. The first runs the Optimizing Brute Force over the set of all control points and QoS levels across all information spaces. The second does the same using the Greedy Approximation algorithm that we developed previously. These algorithms implement a *centralized* QoS management system, with global knowledge of all the information spaces, their control points, and the resources they share. We created these to generate baselines against which to compare our multi-information space algorithms. The centralized Optimizing Brute Force approach will provide us a comparison measure of the best, or most optimal, allocation possible, while the centralized Greedy Approximation provides a comparison measure to the speed and optimality of a centralized algorithm.

Although there are certain scenarios in which a centralized QoS management approach might be acceptable, in general it will not scale as well as the distributed approach, since the performance of the allocation algorithms is based on the number of control points and QoS levels.

Therefore, we are concentrating on the distributed algorithms for our multi-information space QoS management R&D activities, but benchmarking centralized algorithms for comparison and baseline purposes.

We also defined a set of experiments to evaluate the algorithms, and a set of metrics to collect from the experiments. We plan to generate a large enough set of random scenarios (~10,000 with a variety of combinations of available resources) to get sufficiently representative sets of scenarios, ranging from a few shared resources up to many shared resources, with resource scarce to resource rich scenarios, and with few feasible allocations up to many feasible allocations. We will collect metrics about the scenarios to evaluate the coverage and to help group them into subsets, based on metrics such as the percentage of resources shared between information spaces, measures of resource contention, and percentage of feasible solutions.

We plan to run several combinations of first and second phase algorithms and evaluate their performance (speed and optimality of the result) on different subsets of the scenarios.

We continued to investigate strategies for including QMS capabilities as part of the client side libraries of the JBI reference implementation (RI). We have identified three approaches, varying in the amount of capability packaged and in the effort involved in packaging it. We are working on a technical report describing these approaches.

9.21 June 2007

During the report period, we conducted experiments to evaluate the multi-phase resource allocation algorithms for managing resources that are shared across information spaces. As a reminder, the allocation algorithms that we have developed (described in more detail in last month's report) are two-phase algorithms, in which the first phase identifies the resources shared between information spaces and divides them between the information spaces. We have developed three first phase algorithms:

1. *Even Splitter* – Each information space is given an even amount of each shared resource. For example, if two information spaces are sharing a resource X, each would be allocated half of X.

2. *Weighted Splitter* – Each information space is given a share weighted by a factor that can be provided by a commander or computed from the relative priorities of the information spaces.
3. *Dynamic Approximation* – The *Greedy Approximation algorithm* we developed earlier (described in more detail in February 2007's report) is run on a subset of control points, i.e., those that share resources across the information spaces, to calculate the amount of each shared resource allocated to each information space.

The second phase then runs the multiple-resource algorithms we developed earlier (i.e., the *Greedy Approximation* or *Optimizing Brute Force*[13]) within each information space, with each information space only considering its allocation as the total available of the shared resources. For the remainder of this report, we use "approximation" and "greedy" interchangeably to refer to the Greedy Approximation algorithm.

Experimental design. We designed our experiments to enable us to run the algorithms on a large enough number of randomly generated scenarios to ensure that we would have a significant number of scenarios with various characteristics (e.g., small to large amount of resource contention, small to large percentage of feasible allocations, and varying amount of resources shared between information spaces).

The above multi-phase algorithms (six combinations in all, each of the three first phase choices combined with each of the two second phase choices) are the experimental cases. As two baselines against which to compare, we used "centralized" versions of the Optimizing Brute Force and the Greedy Approximation algorithms that we developed previously. The centralized Optimizing Brute Force approach provides a comparison measure of the best, or most optimal, allocation possible, while the centralized Greedy Approximation provides a comparison measure to the optimality and speed of a centralized algorithm.

Our experiments consisted of executing the algorithms on 10,000 randomly generated *scenarios*. Each scenario consists of ten applications, each with three service levels (high, low, and starvation), with each service level utilizing three resources.[14] The scenario generator varies the resources used by each service level, the amount of each resource used by each service level, and the utility associated with each service level.

We conducted the experiments using a simulator that takes each scenario and divides the applications between two information spaces, putting half (i.e., five) in each. It then runs the first-phase algorithms to divide the shared resources followed by the second-phase algorithms for

[13] Both of these algorithms are described in more detail in February 2007's status report. The *subtree-pruning brute force* algorithm walks a tree of applications and service levels to find an optimal solution (i.e., feasible with the highest utility), pruning subtrees that are not feasible or are suboptimal to reduce the search space. The *Greedy Approximation* algorithm maps the multiple-resource allocation problem to a 0-1 integer programming problem and allocates resources using an *effective gradient* ratio of utility divided by resource usage. Subtree-pruning brute force returns an optimal solution but can take exponential time to execute. Greedy approximation is much quicker, but returns an approximate (i.e., not necessarily optimal) solution.

[14] We chose the number of applications (a) and service levels (q) to support running a large number of scenarios against the centralized brute force algorithm baseline, which takes $O(q^a)$ to run. Ten applications with three service levels results in a search space of 59,049 possible allocations. Twenty applications with the same number of service levels drive the search space up to over 3 billion possible allocations.

each information space to choose an allocation. It also runs the two centralized algorithms on each information space to gather the baseline metrics.

The number of applications sharing available resources and the percentage of resources shared across information spaces varied from scenario to scenario, but are affected not only by the factors that the scenario generator varies (i.e., resources used by each service level, the amount of each resource used by each service level, and the utility associated with each service level) but also by the total number of resources available. To attempt to get good representative coverage over a variety of possible scenario configurations, we generated 10,000 scenarios each for a varying number of total available resources: 30, 70, 110, 150, and 190 resources. This results in a total of 50,000 scenarios.

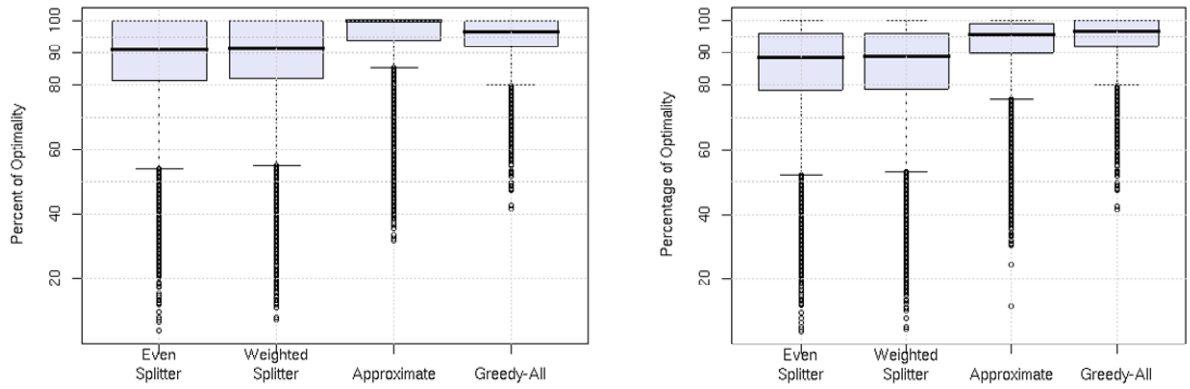
Summary of results. We have generated large amounts of experimental data and are still evaluating and summarizing the results. This status report will present a brief summary of some of the important experimental results and analyses we have performed. We will present a more complete description in a separate report.

Figure 65 illustrates the performance of the six multi-phase algorithms with respect to choosing an allocation, compared to the optimal solution (100% optimality) of the centralized Optimizing Brute Force and the performance of a centralized version of the approximation algorithm (called *Greedy-All* in the figure). All multi-phase algorithms perform well, producing allocations that are a median of nearly 90% or more of optimal. The multi-phase approximation algorithm (running the approximation algorithm during the first and second phases) does nearly as well as the centralized approximation algorithm, and both have medians within a few percentage points of optimal (96% and 97% of optimal, respectively).

Figure 65(a) evaluates the relative contribution to the performance of each of the first phase algorithms. Since this graph uses the Optimizing Brute Force, which produces an optimal allocation, as the second phase of the three multi-phase algorithms, any allocations less than 100% optimal are due to the first phase. The approximation first phase algorithm outperforms the other first phase algorithms by a significant amount, with a median of 100% optimality versus 91% optimality of each of Even Splitter and Weighted Splitter, and with considerably less variance (the majority of solutions are above 80% of optimal).

While the graphs in Figure 65 do show considerably good performance for the multi-phase algorithms, they also show some outliers indicating isolated scenarios in which the algorithms performed poorly. As part of our experimental process, we identified a set of metrics with which we could characterize the scenarios and study how these characteristics impact the optimality of allocations produced by the multi-phase algorithms. The goal is to look for characteristics of scenarios for which the algorithms perform particularly well (close to optimal solutions and no outliers) or particularly poorly (lower utility solutions and more outliers). Some of the metrics that we collected follow:

A. *Percent of feasible solutions:* As described above, each generated scenario has 59,049 possible solutions. Only some of these are *feasible*, i.e., fit within the amount of resources available. The number of infeasible solutions is a good measure of the level of contention in a scenario. That is, the higher the percentage of feasible solutions, the lower the contention for resources in the scenario.



(a) Multi-phase algorithm with various first phase and *brute force (optimal)* second phase compared to centralized brute force (100% optimality) and centralized approximation (Greedy-All).

(b) Multi-phase algorithm with various first phase and *approximation* second phase compared to centralized brute force (100% optimality) and centralized approximation (Greedy-All).

Figure 65: Optimality of the multi-phase algorithms compared to the centralized Optimizing Brute Force (100% optimal) and centralized approximation (labeled as Greedy-All in the graphs).

B. *Highest percent of applications not-starved in any feasible solution:* As contention for a resource increases to the point that there is not enough available to meet the requests of all applications, it is conceivable that only some of the applications requesting a resource will be able to run. The others will be starved (i.e., receive no resource allocation). This metric looks for the feasible solution that runs the highest number of applications (whether that solution is the highest utility solution or not[15]) and collect the number of applications running as a percentage of the number available to run.
C. *Highest percent of applications requesting the most-shared resource:* In general, as more applications request a given resource, the contention for the resource increases. Therefore, as another measure of contention in a given scenario, we compute the largest number of applications requesting a single resource. We collect the number of applications as a percentage of the number available to run.
D. *Highest percent of resource requested by applications (in any service level) requesting the most-shared resource:* This metric looks for the highest amount of a resource requested in any possible allocation. For example, if a scenario has an allocation in which 500% (i.e., 5x of what's available) of a resource is requested and another scenario has no more than 95% of any resource requested, the first scenario indicates a more severe potential bottleneck than the second.

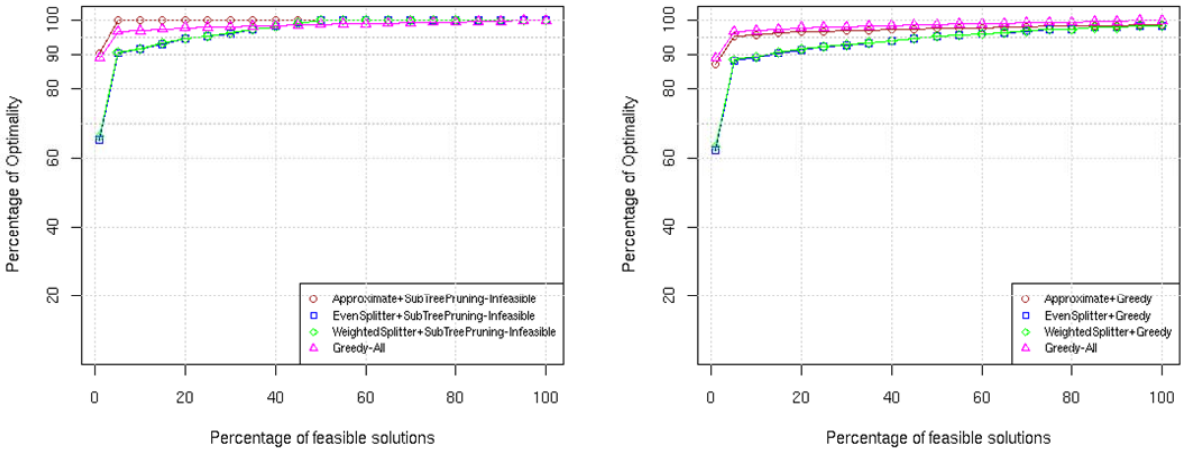
[15] The utility of a solution is based on not only the number of applications that are running but also on the QoS levels at which the applications run, so a solution that runs fewer applications but at higher QoS levels could be of higher overall utility than one that runs more applications but at lower QoS levels.

E. *Percentage of resources shared across the information space:* This metric affects the size of the space over which the first phase algorithms operate. It also indicates a measure of contention for those resources shared between information spaces.

The effects of levels of contention on algorithm optimality. Each metric above measures a different aspect of contention for resources. We analyzed the effect of each of these contention metrics on the solutions provided by the six multi-pass algorithms compared to the optimal solution (provided by the centralized Optimizing Brute Force baseline) and the solution provided by the centralized approximation algorithm. The results show that the two algorithms with the *Dynamic Approximation* first phase perform the best, i.e., Dynamic Approximation-Optimizing Brute Force and Dynamic Approximation-Greedy Approximation, with solutions near optimal under a variety of contention levels and types. The results also show that the choice of second phase has less impact, with the Optimizing Brute Force second phase providing only slightly more optimal solutions.[16]

The following paragraphs and graphs describe the results for the above metrics in more detail.

Figure 66 shows that the multi-phase algorithms perform well even under high levels of contention. The Dynamic Approximation first phase algorithm is clearly the best of the first phase algorithms, performing well with the Optimizing Brute Force and Greedy Approximation second pass. The Dynamic Approximation first phase algorithm with either the Optimizing Brute Force or Greedy Approximation second phase performs very well with even 95% infeasible

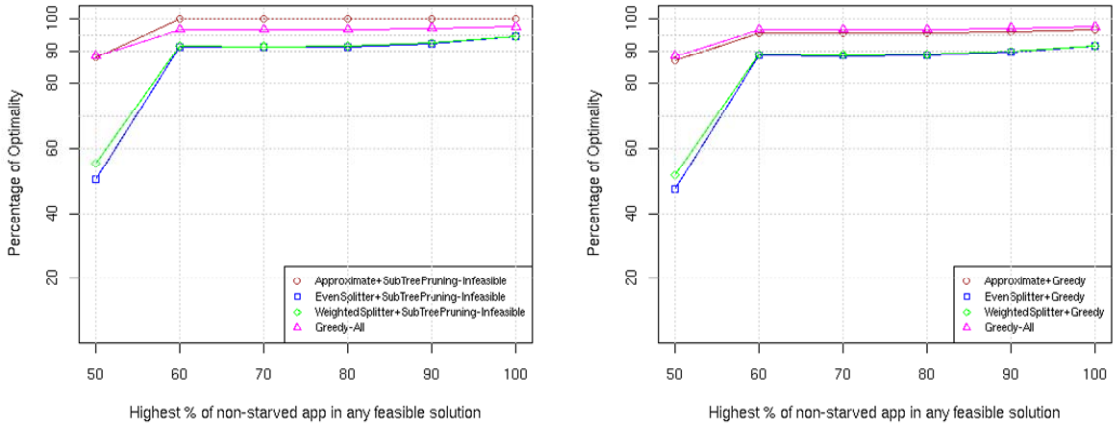


(a) Multi-phase algorithms with various first phase and *brute force (optimal)* second phase

(b) Multi-phase algorithms with various first phase and *approximation* second phase

Figure 66: The effect of the number of feasible solutions (metric A) on the optimality of the multi-phase algorithms.

[16] Our earlier experiments (reported in the February 2007 status report) show the speed comparison of brute force and approximation. While brute force is sufficiently fast for the small information spaces we have in these experiments, approximation scales better providing solutions much faster with larger information spaces, i.e., those with more clients/applications/control points.



(a) Multi-phase algorithms with various first phase and *brute force (optimal)* second phase

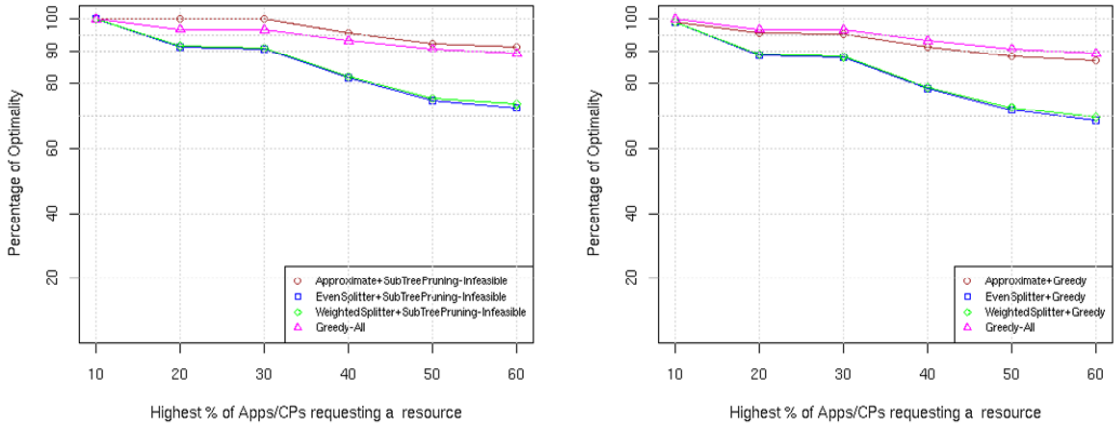
(b) Multi-phase algorithms with various first phase and *approximation* second phase

Figure 67: The effect of application starvation (metric B) on the optimality of the multi-phase algorithms.

solutions (5% feasible solutions), with better than 95% optimality. Even with nearly zero feasible solutions, both multi-phase algorithms with the Dynamic Approximation first phase provide 90% optimal solutions or better.

Figure 67 illustrates the effects of the second contention metric. As would be expected, when feasible solutions don't starve many applications (i.e., the % of non-starved applications is high), the % optimality of the multi-phase algorithms is higher. As Figure 67 shows, when the % of non-starved applications is 60% or above, all of the algorithms provide solutions 90% of optimal or higher. Again, the Dynamic Approximation first phase followed by either the Optimizing Brute Force or Greedy Approximation second phase perform best of all the multi-phase algorithms. Furthermore, they perform significantly better when contention is higher, able to produce solutions with a median of 88%-89% optimality with as much as 50% of applications being starved, and they perform as well or better than the centralized algorithms. In contrast, the median optimality of solutions produced by using an even or weighted split as the first phase declines to 50% in the high contention situations.

As illustrated in Figure , the median percent optimality decreases as the number of applications requesting a bottleneck resource increases. The performance of the multi-phase algorithms using the Dynamic Approximation first phase is still quite good, producing solutions that are close to 90% of optimal (and close to the centralized algorithm) even when as many as 60% of the applications are requesting a single resource. The other first phase algorithms do not fare as well, declining to 72% of optimal as contention for the bottleneck resource increases. Furthermore, the slope of decline in optimality is steeper for the Even and Weighted Splitter algorithms than for the Dynamic Approximation algorithm.

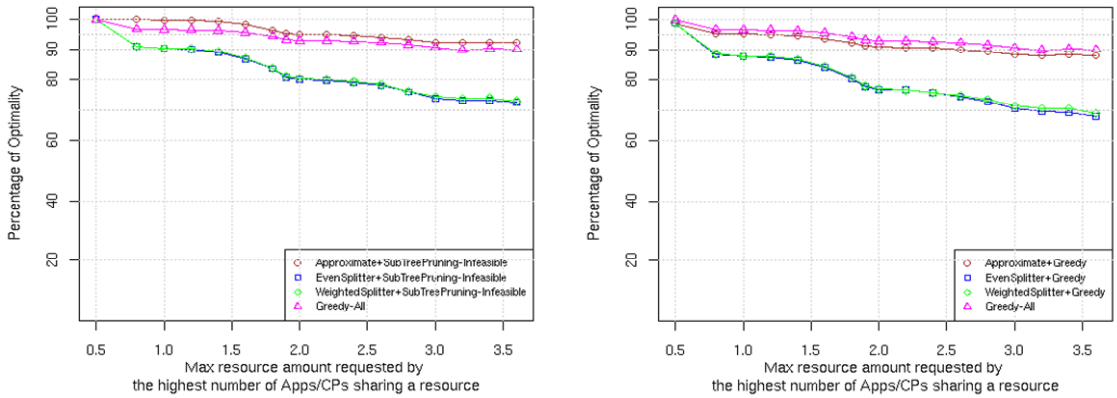


(a) Multi-phase algorithms with various first phase and *brute force (optimal)* second phase

(b) Multi-phase algorithms with various first phase and *approximation* second phase

Figure 68: Percent of optimality comparison of algorithms using metric C – highest percentage of applications requesting the most-shared resource.

While Figure illustrates the effects of a lot of applications requesting a specific resource, it does not take into consideration how much of the resource they collectively request. In contrast, Figure illustrates the effect of the amount of a single, most requested (by quantity requested) resource on the optimality of the algorithms. The *x*-axis in the graphs represent the amount requested of the most requested resource, varying from half of the resource capacity requested (0.5x) to 3.5x more (i.e., 350%) of the resource capacity. As would be expected, when less than the resource capacity (< 1.0) is the most requested, i.e., the scenario represents a resource rich environment, all algorithms produce near optimal solutions. As the amount requested increases, the optimality declines, although the algorithms with the Dynamic Approximation first phase still produce near optimal results and results close to the centralized algorithm, with a median percent optimality above 90%. This particular metric does not seem to affect the outcome of the



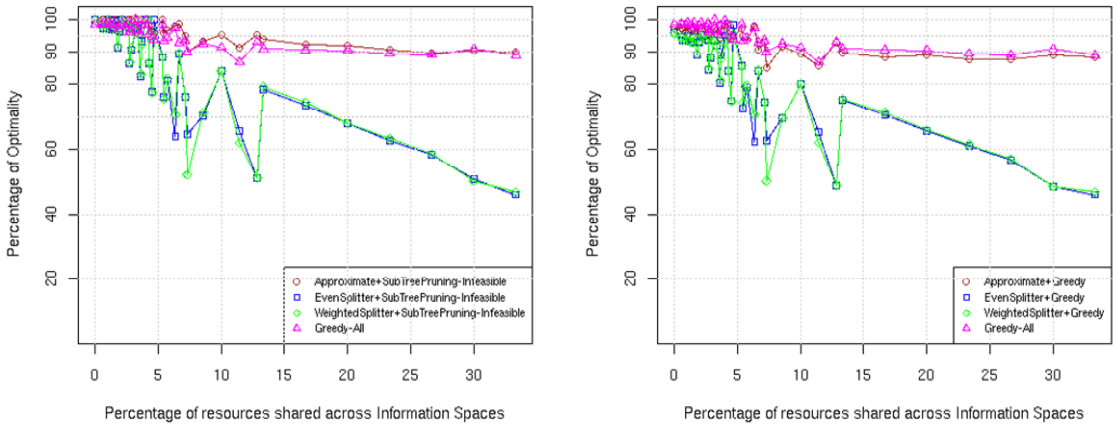
(a) Multi-phase algorithms with various first phase and *brute force (optimal)* second phase

(b) Multi-phase algorithms with various first phase and *approximation* second phase

Figure 69: Percent of optimality comparison of algorithms using metric D – highest amount of resource requested by applications (in any service level) requesting the most-shared resource

second phase algorithm much, as the Dynamic Approximation-Greedy Approximation algorithm (in Figure (b)) performs nearly as well as the Dynamic Approximation-Optimizing Brute Force algorithm (in Figure (a)).

Metric E highlights the differences in efficacy of the choices for the first phase algorithm. As illustrated in Figure 68, as the amount of resources shared between information spaces increases, the median percentage of optimality of the Even Splitter and Weighted Splitter first phase algorithms declines significantly, to as little as 42% of optimal. The two algorithms that use the Dynamic Approximation first phase (Dynamic Approximation-Optimizing Brute Force and Dynamic Approximation-Greedy Approximation) perform very well, with a median percentage of optimality of about 90% and closely tracking the centralized algorithm.



(a) Multi-phase algorithms with various first phase and *brute force (optimal)* second phase

(b) Multi-phase algorithms with various first phase and *approximation* second phase

Figure 68: Percent of optimality comparison of algorithms using metric E – percentage of resources shared across information spaces.

The experiments we have conducted and the initial results and analysis that we have done indicate that the multi-phase algorithm using an Dynamic Approximation first phase and either an Optimizing Brute Force or Greedy Approximation second phase (selected dynamically based on the size of the search space and the deadline for allocation) will produce good (near optimal) QoS solutions in many/most scenarios. We plan to continue to look for the source of outliers and quantify other characteristics of the algorithm (such as runtime performance).

Packaging QMS capabilities as part of the JBI client side libraries. As reported in the last status report, we have begun investigating approaches for including QMS capabilities as part of the client side libraries of the JBI reference implementation (RI). During this report period, we worked on a draft technical approach describing several approaches.

9.22 July 2007

During the report period, we did the following:

- Analyzed the performance evaluation data we had gathered to attempt to identify characteristics of scenarios for which the approximation algorithms perform well (in terms of percent of optimality of the solution) and those for which they don't do as well. We divided the scenarios into two categories: (1) scenarios for which multi-phase algorithms will perform well by solutions that are close to optimal (>= 85% of optimal), and (2) scenarios for which they will not (< 85% of optimal);
- We analyzed the runtime performance of the multi-phase algorithm with our Dynamic Approximation as the first pass and Greedy Approximation as the second pass, and designed experiments to evaluate its runtime performance.

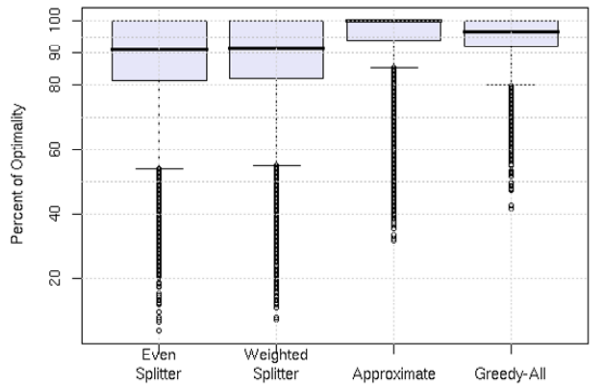
Analysis of the performance evaluation data to categorize scenarios:

In the last report, we described the experiments for evaluating the performance (percent of optimality) of different multi-phase algorithms and reported the results of these experiments. During this period, we attempted to identify specific characteristics or patterns in scenarios in the two categories specified above, i.e., those for which the multi-phase approximation algorithm returns a solution 85% of optimal or better, and those for which it returns a solution less than 85% of optimal. If we could find a characteristic or pattern that would help identify a scenario before running the multi-phase algorithm, it would help us select an algorithm to run for given input scenarios, contentious environments, and deadlines. For example, we could select Dynamic Approximation for scenarios for which we expect to get close to optimal solutions and/or scenarios with a tight deadline to reach a solution. We could select Optimizing Brute Force for scenarios for which we expect the approximation to produce sub-optimal solutions and for which time to reach a solution is less important than reaching a close to optimal solution.

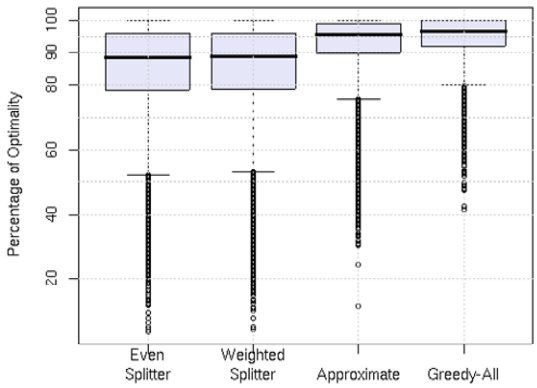
As a reminder, we developed and evaluated six multi-phase resource allocation algorithms for managing resources that are shared across information spaces. These algorithms included a combination of three first phase algorithms: *Even Splitter, Weighted Splitter and Dynamic Approximation,* and two second phase algorithms: *Greedy Approximation and Optimizing Brute Force*. To give a little more context for understanding the work reported in this period, we are including one of the figures (Figure 69) from the last report, with some additional annotation. The figure illustrates the performance of the six multi-phase algorithms with respect to choosing an allocation. Figure 69a depicts the multi-phase algorithm with Optimizing Brute Force as the second phase, and Figure 69b depicts the multi-phase algorithms with Greedy Approximation as the second phase. These algorithms are compared against the baseline of running either the Optimizing Brute Force algorithm that always gives 100% optimality (represented in the figure by 100% on the Y axis) or the Greedy Approximation algorithm (called *Greedy-All* in the figure) in a centralized manner. Here *centralized* means we run the second-phase algorithms to allocate resources for all the applications from all the information spaces. As illustrated in Figure 69, any combination of first-phase algorithm and second pass algorithm performs well, producing an allocation that is a median of nearly 90% or more of optimal.

In this reporting period, we began the analysis by computing the number of outliers[17] that the multi-phase algorithms produced (see Figure 69). From here on, we only discuss the multi-phase

[17] Outliers are defined as values < Q1 – 1.5 * IQR or > Q3 + 1.5 * IQR.



(a) Multi-phase algorithm with various first phase and *brute force (optimal)* second phase compared to centralized brute force (100% optimality) and centralized approximation (Greedy-All).



(b) Multi-phase algorithm with various first phase and *approximation* second phase compared to centralized brute force (100% optimality) and centralized approximation (Greedy-All).

Figure 69: Optimality of the multi-phase algorithms compared to the centralized Optimizing Brute Force (100% optimal) and centralized approximation (labeled as Greedy-All in the graphs).

algorithms that used Dynamic Approximation as the first phase and Greedy Approximation and the Optimizing Brute Force as the second phase. This is because our experimentation results indicate that the Dynamic Approximation outperforms Even Splitter and Weighted Splitter. We found that 9.4% of scenarios produced outliers when Optimizing Brute Force was used for the second phase, and 6.7% of scenarios produced outliers when Greedy Approximation was used for the second phase, from a total of 50,000 scenarios. (Of these outliers, 5.1% were extreme outliers[18] when Optimizing Brute Force was used for the second pass, and 3.5% were extreme outliers when Greedy Approximation was used for the second pass.) Even though the number of extreme outliers is relatively fewer, the total number of outliers motivated us to examine the cause that makes some scenarios produce outliers. Because the graphs presented in the previous report indicated a decline in percentage optimality as levels of contention increased, we started by looking at characteristics that contribute to higher contention, testing the hypothesis that high levels of contention are a cause of sub-optimal allocations.

Recall that we had generated experimental data of scenarios with a varying number of total resources and a fixed number of applications, service levels, and resources in each service level: 10 applications, 3 service levels, and 3 resources per service level chosen from a total varying from of 30, 70, and 110 resources. Each of these experimental data had 10,000 scenarios. These scenarios provide us a good basis for representing different levels of contention. The scenarios with a total of 30 resources were highly contentious as the applications had to choose 3 out of 30 resources. In this case the contention was high because the chance of multiple applications requesting the same 3 resources from a total of 30 was high relative to those that had a total of

[18] Extreme outliers as values < Q1 – 3.0 * IQR or > Q3 + 3.0 * IQR

110 from which to choose 3. As we increased the total number of resources from 30 to 110, the level of contention decreased.

To test the hypothesis, we started examining these scenarios for median percent of optimality, length of (number of scenarios in) the IQR, and the percent of outliers and extreme outliers.

The results when Greedy Approximation is used as the second pass are summarized in Table 5. As expected, we observed that as the level of contention decreased (from 30 total resources to 110 resources), the median percent of optimality increased. The percent of outliers and extreme outliers also increased. We also noticed that as the length of the IQR decreased, meaning that the 50% of solutions from the first to the third quartile were more tightly bunched around the median, the number of outliers and extreme outliers increased. Similar observations were seen when we used Optimizing Brute Force for the second pass. The IQR is a floating window influenced by how tightly grouped 50% of the scenarios are. Hence, an outlier in one set of experiments may not be an outlier in another set of experiments, depending on the IQR window. Therefore, dividing the scenarios solely based on outliers is misleading.

Table 5: Trend in median percent of optimality, IQR, and outliers as the number of total resources increases from 30 to 70 to 110 in the multi-phase algorithm with Greedy as the second pass.

Approx+Greedy	**30 Resources**	**70 Resources**	**110 Resources**
Min % of optimality	11.49425	30.16701	24.48759
Median % of optimality	**88.95260**	**94.66913**	**96.15848**
IQR	**13.62601**	**9.149613**	**7.744562**
1st Outlier	61.14584	75.74697	80.02162
# of Outliers	168	461	642
% of Outliers	**0.336**	**0.922**	**1.284**
1st Extreme Outlier	40.70684	62.02256	68.40478
# of extreme outliers	4	251	495
% of Extreme Outliers	**0.008**	**0.502**	**0.99**

Consequently, instead of focusing solely upon the outliers, we divided the scenarios into two groups: the scenarios that result in >=85% optimality and the scenarios that result in <85% optimality. Then we examined the distribution of scenarios across these two groups for different contentious environments, as defined by the contention metrics listed in Table 6. We observed a significant difference in the values obtained for some of the metrics (percent of feasible solutions, maximum resource requested by the highest percent of applications, percent of resources shared across the information spaces). However, we also noticed a significant overlap in the values for the scenarios for almost all the metrics. The overlap of values for two of the metrics is shown in Figure 70. For example, Table 6 shows a significant gap between the median percent of feasible solutions (metric 1) in the highly optimal and in the sub-optimal categories (42% versus 12%). However, as Figure 70 shows, selecting any scenario and looking at its percentage of feasible solutions, it could fall into either category because of the significant

overlap in their spread. Therefore, these metrics do not appear to be useful for predicting how close to optimal the algorithm will return. For now, we will use the total number of applications, which affects the runtime, to decide whether we want to use Optimizing Brute Force as the second pass or Greedy Approximation as the second pass.

Table 6: Examining metrics as indicators for different categories of scenarios

Approx+Greedy (50,000 total scenarios)	**≥ 85% (45,538 scenarios**	**< 85% (4462 scenarios)**
1. Median % of feasible solutions	42%	12%
2. Highest % of non-starved apps	100%	90%
3. Max % of apps requesting the most shared resources	20%	30%
4. Max resource requested by the highest % of apps	1.5x	1.8x
5. % of resources shared across the infospaces	2%	7%

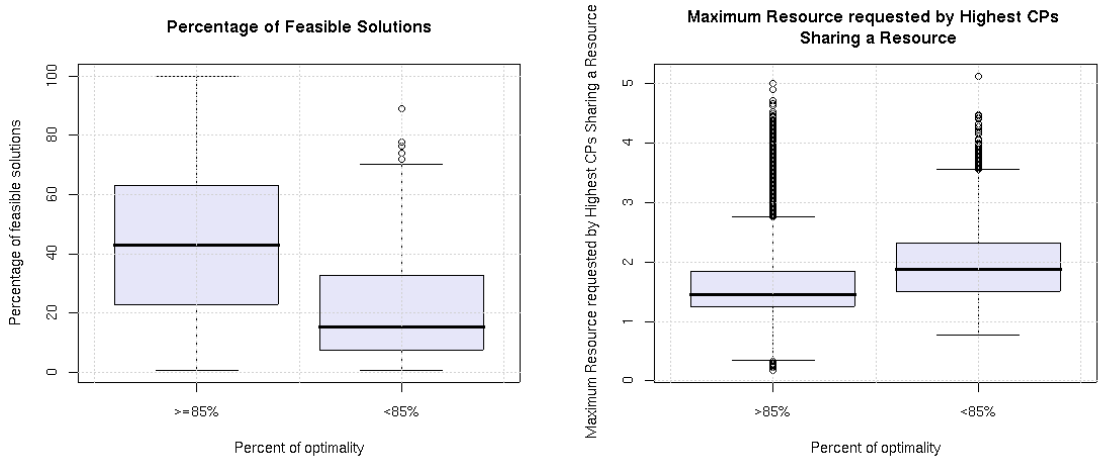


Figure 70: Overlap in two category (>=85% and < 85%) of scenarios for the values for two metrics: percent of feasible solution; and maximum resource request by highest applications requesting the most-shared resource

Theoretical analysis of the multi-phase algorithms (Greedy Approximation as the second phase) for evaluating the runtime and designing experiments to validate the theory:

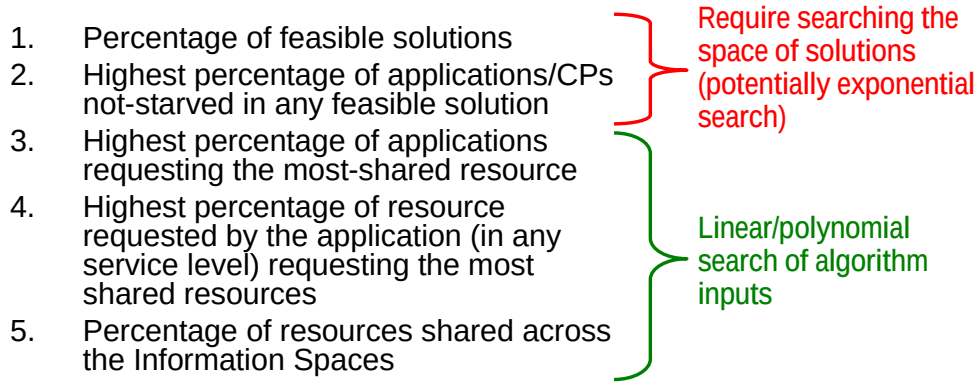
We analyzed the runtime of the multi-phase algorithms with Greedy Approximation as the second phase, and designed experiments to evaluate the runtime of the algorithm by varying the

total number of applications, number of applications shared across the information spaces, and the total number of information spaces. We used only Greedy Approximation as the second phase as the Optimizing Brute Force will result in potentially exponential runtime in the cases where the problem size increases.

As a start, in order to study the impact of contention on runtime, we studied the correlation between the metrics that are obtained by algorithms in linear time, and the metrics obtained by running the Optimixing Brute Force in (potentially) exponential time. Since we will not have access to metrics obtained from the Optimizing Brute Force algorithm in very large scenarios, such as the percent of feasible solutions or the highest percentage of applications not-starved in any feasible solution, the correlation coefficient gives us a way to study the impact of contention on runtime in the cases when we cannot run the Optimixing Brute Force algorithm and hence cannot collect metrics from it. As illustrated in Figure 71, we observed a correlation between the metrics.

Also during the report period, we revised a version of the report on packaging QMS capabilities with the JBI client side libraries. We plan to deliver a copy of this report next month.

We also scheduled a technical interchange meeting to be held during the next report period.



There is a significant correlation between metrics 3-5 and metrics 1-2

Metric	1	2
3	0.6871949	0.6141573
4	0.7157721	0.6309926
5	0.6280571	0.6401662

Figure 71: Correlation coefficients between metrics that are gathered in exponential time and the metrics that are gathered in linear or polynomial time.

9.23 August 2007

During the report period, we analyzed the worst case runtime of some of the multi-phase resource allocation algorithms and conducted experiments to measure their runtime.

As a reminder, in the June report, we described the *multi-phase* algorithms that we developed to support information spaces that share resources. The first phase of the algorithms allocates the resources shared between information spaces, creating constraints on the second phase of the algorithms that allocate resources within each information space. We developed and analyzed three first phase algorithms, *Dynamic Approximation, Even Splitter,* and *Weighted Splitter*, to work with the *Greedy Approximation* and *Optimizing Brute Force* second phase algorithm. This results in six total multi-phase algorithms, from every combination of first- and second-phases:

- Dynamic Approximation-Greedy Approximation
- Even Splitter-Greedy Approximation
- Weighted Splitter-Greedy Approximation
- Dynamic Approximation-Optimizing Brute Force
- Even Splitter- Optimizing Brute Force
- Weighted Splitter- Optimizing Brute Force

Our experiments reported in the June report indicated that the two multi-phase algorithms that used Dynamic Approximation as a first phase outperformed (in percent of optimality) the other algorithms.

During this reporting period, we evaluated the runtime of the *Dynamic Approximation-Greedy Approximation* algorithm. We chose to evaluate the Dynamic Approximation first-phase algorithm because it outperformed the other two first-phase algorithms. We chose to evaluate the Greedy Approximation second-phase algorithm because it runs in polynomial time, in contrast to the Brute Force algorithm which runs in exponential time.

Theoretical analysis of the multi-phase runtime.[19] In previous analysis, we determined that the Greedy Approximation algorithm runs in $O(a^2qr)$, where a is the number of applications, q is the number of QoS levels, and r is the number of resources. The Dynamic Approximation first-phase algorithm runs essentially the Greedy Approximation algorithm, but on a subset of applications, i.e., only those that share resources between information spaces. Therefore, for a set of a applications, split over I information spaces, the Dynamic Approximation first phase takes $O(a'^2qr)$ where $a' \subseteq a$ is the subset of applications that share resources between the information spaces. The Greedy Approximation second phase runs in $max(a_1^2qr, ..., a_I^2qr)$, where a_i is the number of applications in information space i, for $i=1..I$. That is, the second phase takes as long as the largest information space, i.e., the information space with the largest number of applications.

[19] The analysis presented here was performed for the August report and did not include the pass in the Dynamic Approximation algorithm needed to discover the applications that share resources between the information spaces. This pass takes time $ar + aqr$ and is added to the time to run the Dynamic Approximation algorithm presented in the August 2007 monthly report. The runtime analysis in Section 4.5.1.1 includes the time for this pass.

In a representative case in which, say, 10% of the total applications share resources across information spaces and the applications are divided evenly between the information spaces, the runtime would be $O(1/10a^2qr + (a/I)^2qr)$. Which term dominates depends on which is larger, the number of applications sharing resources across the information spaces or the number of applications in each information space.

In the worst case, all of the applications would share resources across information spaces ($a' = a$) and applications would not be evenly spread across the information spaces (one information space could have $a-(I-1)$ applications and each of the rest of the information spaces would have one application each). In this case, the first phase essentially runs a centralized Greedy Approximation algorithm and the second phase is superfluous. The runtime in this worst case is

$$O(a^2qr + (a-I+1)^2qr) = O(2a^2qr) = O(a^2qr)$$

The Dynamic Approximation-Greedy Approximation algorithm runs in polynomial time. In the worst case, the runtime of the Dynamic Approximation-Greedy Approximation algorithm is a constant multiplier of the Greedy Approximation algorithm. However, in the usual case, we expect the runtime of the Dynamic Approximation-Greedy Approximation algorithm to be less than the centralized Greedy Approximation (i.e., *Greedy-All*) algorithm.

Experimental analysis of the multi-phase runtime. Even though our analysis suggests that in the worst case Greedy-All should perform better than Dynamic Approximation-Greedy Approximation, we expect in the general case that the Dynamic Approximation-Greedy Approximation algorithm should run faster than the centralized Greedy-All. This is because we expect that in the usual case a' would be less than a and that applications would be well distributed among information spaces.

We ran experiments to test the runtime of the multi-phase algorithm varying the following variables:

1. The number of total applications
2. The number of resources
3. The level of contention

1. The effect on runtime of varying the total number of applications

The analysis above showed that the number of total applications would significantly impact the runtime of the centralized and multi-phase algorithms. However, we hypothesize that in the usual case where only a subset of applications share resources across information spaces and the total number of applications are well distributed between information spaces, that an increase in the total number of applications would impact the centralized Greedy-All algorithm more significantly than the two phase Dynamic Approximation-Greedy Approximation algorithm. In other words, we expect the multi-phase approximation algorithm to scale better than the centralized version.

We conducted an experiment in which we varied the total number of applications. We used our scenario generator simulation software (described in earlier reports) to generate different scenarios as input to the algorithms. We generated scenarios that varied the total number of

applications from 20 to 2000. We forced a uniform distribution of applications between two information spaces (I=2) by dividing the applications in half, with each information space having half of the total number of applications (i.e., 10 to 1000 applications each). We set the scenario generator parameters to 3 QoS levels per application, 3 resources used per QoS level, and 110 total number of resources. The scenario generator randomly generated the utility value for each QoS level, the specific resources (from the 110 available) used by each QoS level, and the amount of each resource used. The choice of 110 available resources (a fairly large number compared to the number of resources, 3, used by each application) makes it likely that the number of applications sharing resources across the information spaces is a *subset* of the total number of applications (i.e., the more available resources to choose from, the less likely that two applications will share resources).

We generated 100 random scenarios for each value of the number of applications, ranging from 20 to 2000, and plotted the median algorithm runtime for each set of scenarios, as displayed in Figure 72.

As the total number of applications increases, the median runtime of both the Dynamic Approximation-Greedy Approximation algorithm (shown with blue line and diamonds) and Greedy-All algorithm (shown with red line and squares) increase. The centralized Greedy-All algorithm outperforms the multi-phase algorithm slightly at a low number of total applications (up to a few hundred per information space). As the number of applications increase beyond that, the runtime of the multi-phase algorithm increases at a much slower rate than the centralized algorithm, indicating that the multi-phase algorithm scales better in terms of number of total applications.

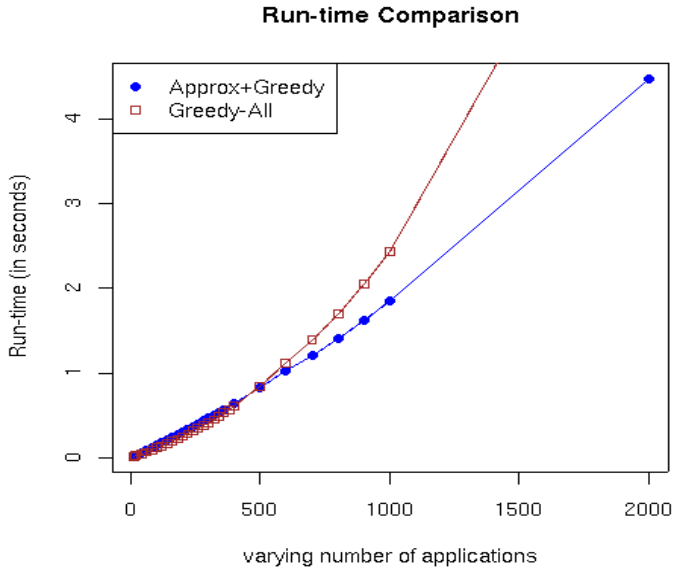


Figure 72: Median runtime of Dynamic Approximation-Greedy Approximation (Approx+Greedy) when we varied the number of applications from 20 to 2000. The centralized approximation algorithm, Greedy-All, is shown as the baseline.

2. The effect on runtime of varying the total number of resources

Because of the linear contribution of the *r* term to equation (1), we hypothesize that the runtime of Dynamic Approximation-Greedy Approximation should change linearly as the total number of resources changes.

We tested this hypothesis by conducting an experiment in which we varied the total number of resources. As in the experiments above, we used the scenario generator simulation software to generate random scenarios varying the total number of resources from 10 to 150. We set the scenario generator parameters to 3 QoS levels per application, 3 resources used per QoS level, and 50 applications. We divided the applications into two information spaces (I=2) such that each information space had half of the total number of applications (i.e., 25 applications). We used 5000 scenarios for each experimental data set and plotted the results as the boxplots in Figure 73. The results support the hypothesis as the graph indicates an approximate linear slope for the median (thick dark line) runtime as the number of resources increases.

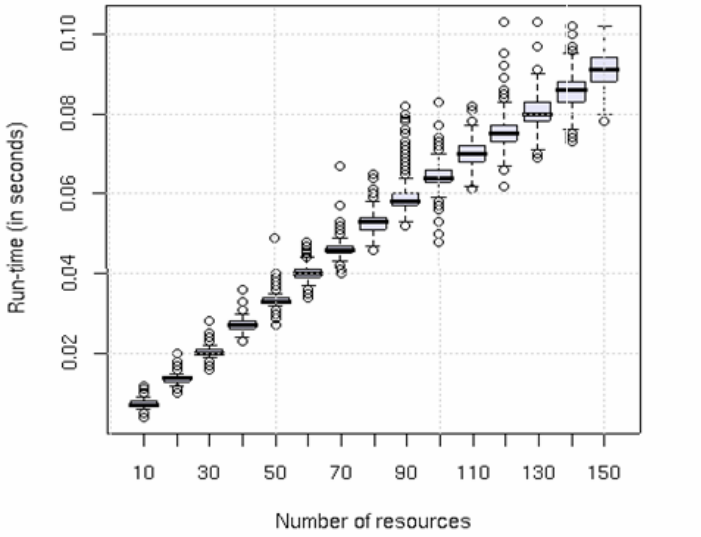
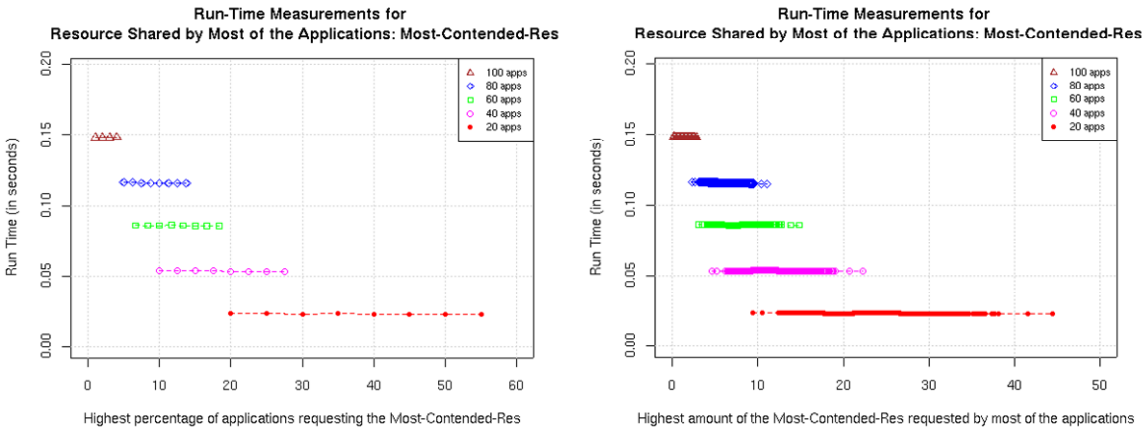


Figure 73: Runtime of Dynamic Approximation-Greedy Approximation as a function of varying the number of resources

3. The effect on runtime of varying the level of contention

We measure the level of contention using the following metrics, described in more detail in the June status report:

- Percentage of applications requesting the most-shared resource.
- Maximum amount requested of the most-shared resource.
- Percentage of resources shared across the information space.



(a) Runtime of Dynamic Approximation-Greedy Approximation as a function of the highest percentage of applications requesting the most-shared resource

(b) Runtime of Dynamic Approximation-Greedy Approximation as a function of the maximum amount requested of the most-shared resource

Figure 74: Runtime of the Dynamic Approximation-Greedy Approximation algorithm plotted against the level of contention in the scenarios.

Our previous experiments (reported in June) indicate that level of contention affects the *efficacy* of the algorithms in terms of percent of optimality of the solution produced, but based on our analysis we do not expect the level of contention to affect the runtime *efficiency* of the algorithms much, if at all.

We tested this hypothesis by conducting an experiment in which we generated a large number of random scenarios[20] with various numbers of applications, with the expectation that the generated scenarios would cover a spectrum of levels of contention. We ran the Dynamic Approximation-Greedy Approximation algorithm on each scenario and plotted a regression line for the runtime against the level of contention.

Figure 74 illustrates that the level of contention as defined by the first and second metrics do not significantly affect the algorithm runtime. Figure 74a shows that as the percentage of applications requesting the most shared resource increases, the runtime remains approximately unchanged. Likewise, Figure 74b shows that as the percentage amount requested of the most requested resource increases, the runtime also remains approximately unchanged.

The declining ranges of the plotted measurements indicate that as the number of applications increases, the percentage requesting any specific resource declines. Though there are the same number of scenarios for the larger number of applications, the level of contention falls within a significantly narrower range. Unfortunately, this means that the generated scenarios at those higher numbers of applications represent a lower variety of contention possibilities, and therefore their regression lines are less compelling for determining the effect of contention on runtime. However, the full set of scenarios and the obvious lack of increase in runtime even at the higher

[20] 5000 random scenarios each with the number of applications varying from 10 to 100 (in steps of 10). The scenario generator parameters were set to 3 QoS levels per application, 3 resources used per QoS level, and 110 total resources.

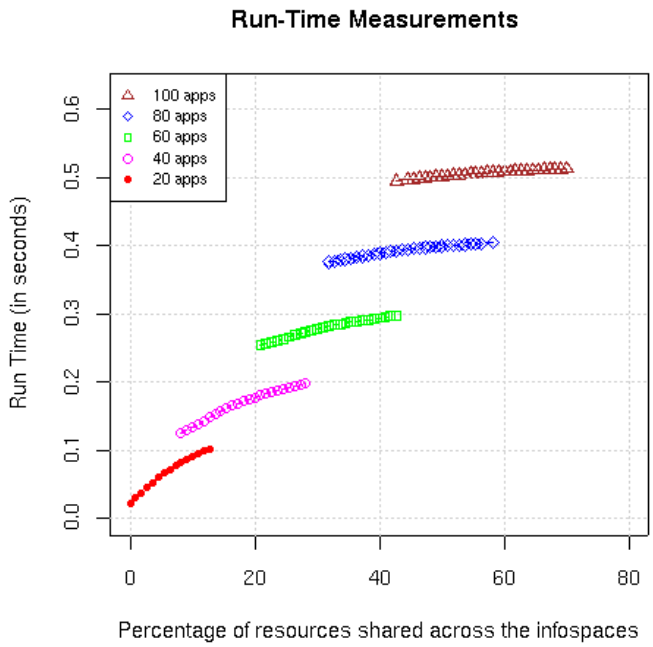


Figure 75: Runtime of Dynamic Approximation-Greedy Approximation as a function of the percentage of resources shared across information spaces

range of contention of the lower regression lines provides evidence to support the hypothesis that contention does not affect runtime as much as other factors.

Figure 75 illustrates the impact of an increase in the percentage of resources shared across information spaces. As this metric increases, it affects the runtime of the first phase (the space of applications the first phase runs over gets larger). As the number of total applications increases, the second phase comes to dominate the algorithm's runtime and the impact of an increase in the number of shared resources impacts the runtime by a lesser amount.

In summary, the analysis and experiments that we are describing during this report period show the following results:

- The multi-phase approximation algorithm runs in polynomial time with a worst case runtime that is a constant multiplier of the runtime of the centralized approximation algorithm.
- The multi-phase approximation algorithm scales better than the centralized algorithm with respect to number of applications.
- Increasing the number of total resources affects the runtime of the multi-phase algorithm linearly.
- Changes in the level of contention (as defined by the number of applications requesting a most shared resource and the maximum amount of that resource requested) have less effect on the runtime of the multi-phase algorithm than other factors.

- The amount of resources shared across information spaces affects the runtime (approximately linearly) but has more effect when information spaces are small (containing fewer applications) than when they are large (containing more applications).

Host a technical interchange meeting. On August 16, 2007 we hosted a TIM at BBN, Cambridge, MA. At the TIM, we presented a summary of progress to date and the progress since the last TIM, including the multi-phase QoS algorithms that we have developed for allocating QoS across information spaces and the efficacy evaluations of the multi-phase algorithms. We also described approaches toward integrating our QoS Management System (QMS) into the JBI client-side libraries, and delivered a draft report describing the approaches.

Begin documenting the results of the experiments. During the reporting period, we began a report summarizing the design of QoS management algorithms and their experimental evaluations.

9.24 September 2007

During the report period, we continued working on a report documenting the results of the experiments we conducted on the QoS allocation algorithms, began designing the next version of the QMS utilizing these algorithms, began planning our final demonstration, and created a poster for the upcoming OIM SAB review.

Documentation of the experimental results. In the previous status reports, we described experiments that we conducted on our multi-resource multi-QoS allocation algorithms, evaluating their effectiveness (percent of optimality) in allocating QoS within and across information spaces, their runtime, and the effect of resource contention on their effectiveness and runtime. During the report period, we continued producing a report documenting the results of these experiments.

Designing a multi-resource multi-QoS QMS prototype. During the report period, we began designing the next version of the QMS components utilizing the new QoS allocation algorithms we described in previous reports. The new algorithms allocate QoS levels and resources to applications sharing resources within or across information spaces. Based on this design, we have begun prototyping these QMS components.

Planning the final demonstration. During the report period, we also began planning the final demonstration. Two possibilities for a demonstration are one based on our previous demonstrations, but using the new algorithms, and one based on our simulator that we utilized to conduct our experiments. The former possibility would build upon our previous demonstrations and show some operational relevance, but would necessarily be limited in scope because of the size of our demonstration testbed (which would limit us to only a few information spaces containing a few clients). The latter possibility would support a demonstration of larger scale with more simulated parts, but would lack some of the operational realism of the former. Our current plan is to take the second approach, in order to better showcase the new capabilities of the QMS system.

SAB poster. In support of the OIM review before the Scientific Advisory Board (SAB), we created a poster documenting our ICED and Context Aware QoS results, and our planned QED

activities. We delivered the draft poster to AFRL on September 18 and a revised version on September 25. We will present the contents of this poster at the SAB review in October.

9.25 October 2007

During the report period, we completed documenting the results of the experiments we conducted on the QoS allocation algorithms. These experiments, described in previous status reports, evaluated our algorithms' effectiveness (percent of optimality) in allocating QoS within and across information spaces, their runtime, and the effect of resource contention on their effectiveness and runtime. The report that we produced was uploaded to Jiffy on October 31.

Also during the report period, we continued designing and began implementing the next version of the QMS components utilizing the new QoS allocation algorithms. We re-architected the SRM component of the QMS to use the new algorithms, to separate the algorithmic part of the SRM from the other parts, and to operate in a multi-threaded, distributed configuration. The resulting SRM allows multi-phase allocation algorithms to be plugged in and chosen between at runtime. For our demonstration, we prototyped one first phase algorithm (Dynamic Approximation) and two second phase algorithms (Greedy Approximation and Optimizing Brute Force). We also developed interfaces and functionality so the SRM handles the synchronization of policy dissemination and temporary disconnections. We also made the main part of the SRM that invokes the allocation algorithm thread-safe. With these new capabilities, the SRM allocates QoS levels and resources to applications sharing resources within or across information spaces.

Also during the report period, we began planning for the final review of the DynRIIC project, scheduled for November 15 at AFRL. We began working on a set of slides for the final presentation, plans for packaging and delivering the final software and the final users' guide and demonstration script.

As part of planning for the final review, we began developing the final demonstration to show at the final review. The demonstration will showcase the QoS allocation capabilities of the SRM component with the new QoS management algorithms, including the ability to allocate QoS levels and resources to applications within and between information spaces and to scale to large numbers of applications across multiple information spaces. The demonstration will also show the selection of algorithms, the optimality of QoS allocations, and the execution time of the algorithm. In addition to developing the enhanced SRM described above, we also began implementing a demonstration driver version of our Mission Manager component; a new version of our QoS internals' display that displays the operation, effectiveness, and execution time of the SRM; and components to provide the QoS levels, utility, and resource usage input that the SRM needs.

In conjunction with building the enhanced SRM and the demonstration software, we began working on an updated Users' Guide and Demonstration Script to accompany the demonstration.

We supported the review of the OIM projects before the Scientific Advisory Board (SAB). We conducted a dry run of our poster presentation early in the month. On October 15, we attended the SAB review and presented our poster documenting our ICED and Context Aware QoS results, and our planned QED activities before the SAB board.

We attended the Minnowbrook planning meeting from October 22-25 and led the session on QoS Enabled Dissemination. We prepared a set of outbrief slides documenting the discussion and conclusions of the session and distributed them to attendees on October 29.

9.26 November 2007

During the report period, we wrapped up technical efforts on this project.

This included prototyping an enhanced version of the QMS SRM component with the new QoS management algorithms that we reported on in previous status reports. It also included developing the support software needed to build and run the SRM. We completed an updated version of the Users' Guide (version 2.0) documenting the capabilities and use of the new SRM.

We also developed the software for a demonstration of the enhanced SRM, including displays, configuration files, and demonstration script files. We designed a demonstration script and documented it in an updated version of the DynRIIC Demonstration Script document (version 2.0).

We packaged the new elements of the QMS software, demonstration software, and the Users' Guide and Demonstration Script for a final release. This preparation included:

- Packaging the software
- Writing make files and scripts to facilitate ease of building and deploying the software
- Testing the software to iron out as many bugs as possible.
- Burning DVDs for easily handing over the released software.
- Completing the Users' Guide and Demonstration Script.

We conducted the final review at Rome Research Center on November 15, 2007. At this review, we presented a summary of the major results achieved in the DynRIIC project, demonstrated the enhanced SRM software, and delivered the final version of the DynRIIC prototype software and the Users' Guide and Demonstration Script version 2.0 document.

References

1. BBN Technologies, Dynamic Reconfiguration and Interoperation in Infospace Communities User's Guide/Demonstration Script version 1.0., September 28, 2006.
2. BBN Technologies, Dynamic Reconfiguration and Interoperation in Infospace Communities User's Guide/Demonstration Script version 2.0., November 15, 2007.
3. Boxplots: www.wikipedia.org/Boxplots
4. V. Combs, R. Hillman, M. Muccio, and R.McKeel. Joint Battlespace Infosphere: information management within a C2 enterprise. The Tenth International Command and Control Technology Symposium (ICCRTS), 2005.
5. Component Integrated ACE ORB (CIAO), http://www.cs.wustl.edu/~schmidt/CIAO.html
6. E. Eide, T. Stack, J. Regehr, and J. Lepreau. Dynamic CPU management for real-time, middleware-based systems. Proceedings of the Tenth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, ON, May 2004.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
8. Leslie Hall, Computational Complexity, The Johns Hopkins University, http://www.esi2.us.es/~mbilbao/complexi.htm.
9. IETF, *An Architecture for Differentiated Services*, http://www.ietf.org/rfc/rfc2475.txt
10. Infospace Concept Exploration and Development BAA, BAA-05-03-IFKA.
11. Chen Lee, John Lehoczky, Ragunathan Rajkumar, and Dan Siewiork. On Quality of Service Optimization with Discrete QoS Options. Fifth IEEE Real-Time Technology and Applications Symposium (RTAS'99).
12. M. Linderman, B. Siegel, D. Ouellet, J. Brichacek, S. Haines, G. Chase, and J. O'May. A reference model for information management to support coalition information sharing needs. The Tenth International Command and Control Technology Symposium (ICCRTS), 2005.
13. J. Loyall, R. Schantz, D. Corman, J. Paunicka, and S. Fernandez. A distributed real-time embedded application for surveillance, detection, and tracking of time critical targets. Real-time and Embedded Technology and Applications Symposium (RTAS), San Francisco, CA, March 7-10 2005.
14. J. Loyall, P. Sharma, M. Gillen, and R. Schantz. A QoS management system for dynamically interoperating net-centric systems. Proceedings of the SPIE Conference on Defense Transformation and Net-Centric Systems, Orlando, FL, April 9-12, 2007.
15. P. Manghwani, J. Loyall, P. Sharma, M. Gillen, and J. Ye. End-to-end quality of service management for distributed real-time embedded applications. The Thirteenth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2005), Denver, Colorado, April 4-5, 2005.
16. Silvano Martello, Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.
17. Object Management Group, *CORBA Component Model*, V3.0 formal specification, http://www.omg.org/technology/documents/formal/components.htm

18. Object Management Group, *Data Distribution Service for Real-time Systems*, Version 1.2, formal/07-01-01, January 2007.
19. Object Management Group, *Notification Service Specification*, Version 1.1, formal/04-10-11, October 2004.
20. OpenDDS, http://www.opendds.org/
21. R–package: http://www.r-project.org/
22. Rgl library: http://rgl.neoscientists.org/gallery.shtml
23. Richard E. Schantz, Joseph P. Loyall, Michael Atighetchi, and Partha Pal. Packaging quality of service control behaviors for reuse. ISORC 2002, The 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing, Washington, DC, April 29 - May 1, 2002.
24. P. Sharma, J. Loyall, G. Heineman, R. Schantz, R. Shapiro, and G. Duzan. Component-based dynamic QoS adaptations in distributed real-time and embedded systems. International Symposium on Distributed Objects and Applications (DOA), Agia Napa, Cyprus, October 25-29, 2004.
25. Praveen Sharma, Joe Loyall, and Matthew Gillen. Multi-resource multi-QoS algorithms for individual and multiple information spaces, report on efficacy and efficiency experiments, 2007.
26. Y. Toyoda. A simplified algorithm for obtaining approximate solution to zero-one programming problems. *Management Science*, 21, 1975.
27. U.S. Air Force. A guide for communities of interest (COIs), implementing the DoD net-centric data strategy and the Air Force information and data management strategy, Version 1.0, April 2005.
28. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource reservation protocol. *IEEE Network*, 7(6), September 1993.

List of Acronyms

AC	Asset Communicator
AFRL	Air Force Research Laboratory
AOI	Area of Interest
BDA	Battle Damage Assessment
C2	Command and Control
CAPI	Common Application Program Interface
CCM	CORBA Component Model
CIAO	Component Integrated ACE ORB
CM	Connectivity Monitor
COA	Community of Action
COI	Community of Interest
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DDS	Data Distribution Service
ECP	Engineering Change Proposal
EO	Electro-Optical
DynRIIC	Dynamic Reconfiguration and Interoperation in Infospace Communities
GAH	Greedy Achilles Heel
GUI	Graphical User Interface
ICED	Infospace Concept Exploration and Development
IDL	Interface Description Language
IMS	Information Management System
Infospace	Information Space
IQR	Interquartile Range
ISR	Intelligence, Surveillance, and Reconnaissance
JBI	Joint Battlespace Infosphere
LRM	Local Resource Manager
MMC	Mission Manager Coordinator
MMKP	Multi-Resource Multi-QoS Knapsack Approximation
MRMQ	Multi-Resource Multi-QoS
NRA	Non-Role Appropriate
OIM	Operational Information Management
PI	Principal Investigator
QED	QoS Enabled Dissemination
QMS	QoS Management System
QoS	Quality of Service
RAM	Random Access Memory
RI	Reference Implementation
SAB	Scientific Advisory Board
SRM	System Resource Manager
TAO	The ACE ORB
TID	Tactical Information Dominance

TIM	Technical Interchange Meeting
TST	Time Sensitive Targeting
TT	Target Tracking
UAV	Unmanned Aerial Vehicles
XML	Extensible Markup Language